1974

A microprogrammed I/O interface.

Duarte, Raimundo Nonato Daniel

Monterey, California. Naval Postgraduate School

A MICROPROGRAMMED I/O INTERFACE

Raimundo Nonato Daniel Duarte

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MICROPROGRAMMED I/O INTERFACE

by

Raimundo Nonato Daniel Duarte

Thesis Advisor: Raymond H. Brubaker

March 1974

Approved for public release; distribution unlimited.

A MICROPROGRAMMED I/O INTERFACE

by

RAIMUNDO NONATO DANIEL DUARTE
LIEUTENANT - BRAZILIAN NAVY

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1974

ABSTRACT

This thesis presents a basic hardware model suitable for most sequential microprogrammed devices. A software system is described which allows the use of an assembly-level programming language instead of the binary representation of microcodes. The implementation of a microprogrammed input/output interface is presented as an example of use of both the hardware and software.

TABLE OF CONTENTS

LIST OF DRAWINGS

I. INTRODUCTION

This thesis is part of a larger effort to implement a communications network for present and future computer systems at the Naval Postgraduate School. Microcomputers will be used in this network to replace as many interface hardware functions as possible with software, thus providing a degree of flexibility not attainable with hardware-only configurations. The need arose for a device which allows exchange of data and control signals between any of the computer systems and its associated microcomputer.

The aim of this thesis is to develop basic hardware that can be used in any of these interfaces, as well as in most sequential devices.

The IBM System/360 interface was chosen as the guide for design for the following reasons:

a) it has a standard I/O interface between the data channel and the control units which activate I/O devices;

b) it is possibly one of the more complex interfaces, thus providing a worst-case design.

During the course of work the need for a microprogramming language was recognized; the software designed to support it is described in chapter V.

II. IBM SYSTEM/360 I/O INTERFACE

A. OVERVIEW

Whenever the IBM System/360 channel wants to receive/send information from/to a specific I/O device it sends a command (Read/Write) to the device via its control unit and logically disconnects as soon as the control unit acknowledges the command. When the I/O device is ready to send/receive the desired information it signals to the channel which executes a polling sequence to find out which unit is asking for service. If the control unit is busy and cannot accept the command, a "Control Unit Busy Sequence" takes place, whereby the channel is notified and defers its request for a later point in time.

The control unit can also initiate a data exchange by signalling to the channel and waiting until it is ready to service the request.

Due to the number of signalling lines used, the detailed operational description is quite involved. It is described in Ref. 1. Reference 2 contains a somewhat more detailed and readable explanation of some of the different sequences.

B. INTERFACE FUNCTIONS

The rules which constitute the I/O Interface are physically implemented by 34 wires, or lines, whose state can be either up (one, high) or down (zero, low).

The lines are :

Bus Out - a set of nine lines used to transmit information (data, I/O device address, commands) from the channel to the control units. Eight lines are used to convey the information itself and one line is a parity bit. The type of information transmitted over Bus Out is indicated by the state of other lines.

Bus In - a set of nine lines used to transmit information (data, I/O device identification, status information) from the control unit to the channel. Eight lines are used to convey the information itself and one line is a parity bit. The type of information transmitted over Bus In is indicated by the state of other lines.

Address In (abbreviated AdrIn) - is a line from all attached control units to the channel. Its rise indicates that the address of the currently selected I/O device is available on BusIn .

Status In (abbreviated StaIn) - is a line from all attached control units to the channel. Its rise indicates that the control unit has placed status information on BusIn.

Service in (abbreviated SerIn) - is a line from all attached control units to the channel. Its rise indicates to the channel that the selected I/O device wants to transmit or receive a byte of information.

Command Out (abbreviated ComOut) - is a line from the channel to all attached control units. Its rise may indicate:

1) after the rise of AdrIn - the contents of BusOut is a command.

2) after the rise of SerIn - the channel is ending the current operation.

3) after the rise of StaIn - the control unit should disconnect from the interface after the fall of SelOut.

Service Out (abbreviated SerOut) - is a line from the channel to all attached control units. Its rise indicates to the selected I/O device that the channel has accepted the information on BusIn or has provided on BusOut the data requested by SerIn.

Suppress Out (abbreviated SupOut) - is a line from the channel to all attached control units and is used both alone and in conjunction with other outbound lines to provide the following special functions:

1) data suppression,
2) status suppression,
3) command chaining and
4) selective reset.

These functions are described in Ref. 1.

Operational Out (abbreviated OplOut) is a line from the channel to all attached control units and is used for interlocking purposes. Except for SupOut all lines from the channel are significant only when OplOut is up. Whenever OplOut is down, all inbound lines from the control units must drop and any operation currently in process must be reset.

Operational In (abbreviated OplIn) - is a line from all attached control units to the channel and is used to signal to the channel that an I/o device has been selected.

Select Out (abbreviated SelOut) - SelOut and SelIn form a closed loop from the channel through all attached control units and back to the channel.

Select In (abbreviated SelIn) - is the name given to SelOut when it reaches the channel after passing through all control units.

Hold Out (abbreviated HoldOut) - is a line from the channel to all attached control units and is used in conjunction with SelOut.

Address Out (abbreviated AdrOut) - is a line from the channel to all attached control units. It provides two functions:

1. I/O Device Selection - AdrOut up is an order to all attached control units to decode the I/O device address on BusOut.

2. Disconnect Operation - whenever HoldOut is down and AdrOut rises, or AdrOut is up and Hold Out falls, the presently connected control unit must drop OplIn, thus disconnecting from the interface.

Request In (abbreviated ReqIn) - is a line from all attached control units to the channel. Its rise indicates that a control unit is requesting a selection sequence.

Metering Out - is a line from the channel to all attached control units. Its rise indicates that the CPU meter is recording time.

Clock Out - is a line from the channel to all attached control units. Control units should not be allowed to switch from "On-line" to "Off-line" condition when ClockOut is up.

The functions implied by the list above were to be implemented, resulting in the design of a device capable of acting as a control unit.

III. THE APPROACH

An interface to the /360 channel certainly had to include some logical circuitry. Preliminary studies showed that the state of the lines alone is not always sufficient to decide the action to be taken by the device. Therefore the nature of the functions to be performed was not strictly combinational, and the device would have to keep track of event sequences.

Another difficulty was that the number of variables involved, even reducing the problem to the bare essentials, was around seven; this implied the use of large reduction maps, difficult to visualize and error-inducing. The needed addition of flip-flop counters to make up for the sequential nature of some of the functions would aggravate the problem.

Furthermore, a troublesome and time-consuming implementation phase was anticipated for the design. If patchboards were to be used in the experimental implementation, poor contacts and misrouted wires were likely to compound with design errors; on the other hand, hardwired prototyping would be expensive if several corrections or changes were to be made.

These factors led to the use of microprogramming as opposed to hardwiring (or random logic).

IV. MICROPROGRAMMING

A. INTRODUCTION

Microprogrammming, as used in this report, is a design technique substitute to hardwiring. The fundamental idea behind microprogramming is that, given a truth table with n inputs and one output, we can think of it as being a table of contents of a 2^n word, one bit per word, storage device. The state of the inputs determines one unique address in the storage device and the content of this location is the desired value of the function.

It is easily seen that if, instead of one-bit words, the store had, say, eight-bit words, eight separate switching (binary) functions could be implemented. In the application described here, several binary function values are grouped into a field to specify one of several values. For example, a field of three bits can take eight different values. The same table can simultaneously implement several such functions.

The need was for a device capable of implementing the following basic flowchart operations:

1) Conditional branch - where the decision variable was to be one of the I/O interface lines.

2) Unconditional branch.

3) Execute predefined process - where the "predefined process" would be of the form "RAISE LINE..." or "DROP LINE..." only.

B. BASIC HARDWARE

Before introducing the complete model, its basic components will be presented and briefly explained.

1) Read Only Memory (ROM) (Figure 1) - depicted in the diagrams as a rectangle divided in three rows; the bottom row represents the input section and contains a description of the physical device as well as the input (address) bits. The middle row is subdivided in three fields :

leftmost field is the 'next basic address field' or ADR

center field is the 'select field' or SEL

rightmost field is the 'opcode field' or OPCODE

The upper row is subdivided in as many squares as the number of bits in each word of the ROM. The number inside the squares represent the significance of the bit (i.e. the binary order) .

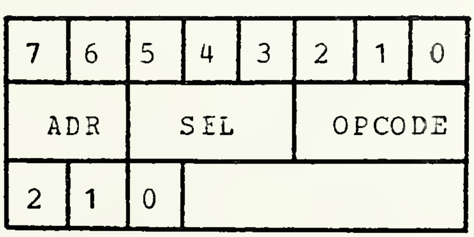


Figure 1. An eight word, eight bits per word, Read Only Memory.

2) Clock - depicted as the Greek letter phi(ϕ). Subscripts are used to differentiate among phases of the clock, i.e. $\phi 1$, $\phi 2$, ... ϕn are all pulse generators with the same frequency; however, the leading edge of the pulse which produces changes in the circuits under their control occurs at distinct time instants.

3) D flip-flops (Figure 2) - depicted as a square with the letter 'D' inside and subscripts whenever necessary to differentiate among the various flip-flops. Whenever the clock rises the output of the flip-flops becomes equal to the input value immediately prior to the clock pulse.

4) Data Selector / Multiplexer (MX) (Figure 2) - logic circuit with 2^n input lines, n select lines and one output line. It is the logical equivalent of a single-pole, 2^n-position switch whose position is specified by a n-bit input address. The output line presents the value of the single input line selected by the select lines. In addition to the input and select lines, the multiplexer has a strobe or enable line. The output is valid only when the strobe line is zero (low).

5) Decoder / Demultiplexer (DMX) (Figure 2) - logic circuit with n inputs and 2^n outputs. For each binary value at the input, one different output line is dropped. In addition, the demultiplexer has a strobe or enable line. The selected output changes state only when the strobe line is low (zero).

C. OPERATION

The operation of the model is better explained by an example. The following assumptions are made:

1) The hardware configuration is as depicted in Figure 2.

2) The circuit is in steady-state operation.

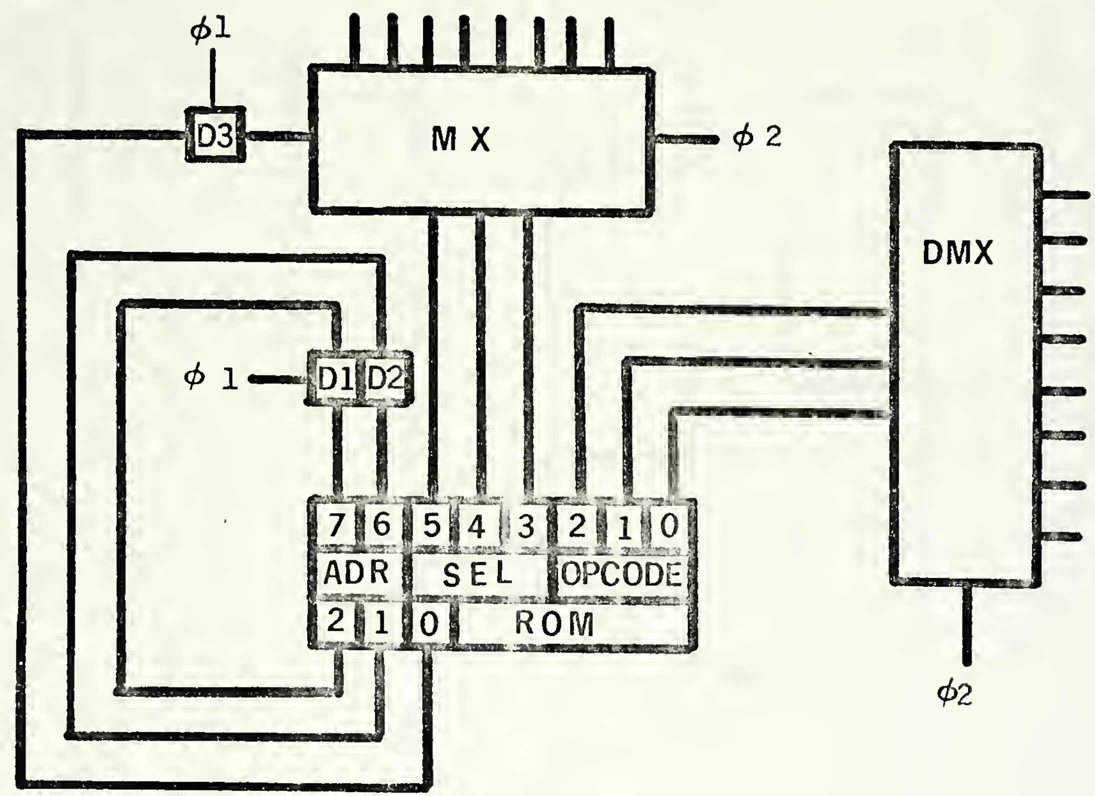


Figure 2. Basic hardware

3) The ROM has already been programmed and the contents of some addresses are tabulated below:

TABLE I

ADDRESS	ADR FIELD	SEL FIELD	OPCODE FIELD
3	10	010	000
4	00	001	011
5	01	001	001
0	01	000	000
1	00	000	000

4) The two clocks ($\phi 1$ and $\phi 2$) run at, say, 500 KHz, their phase relationship being as shown in figure 3 .

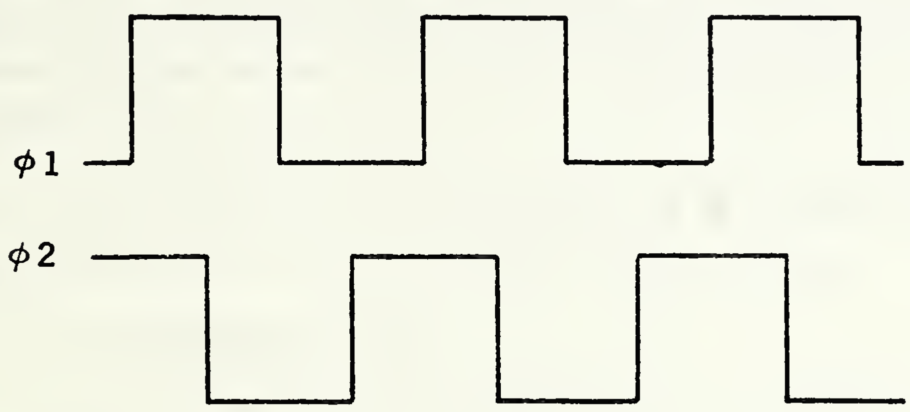


Figure 3. Phase relation between $\phi 1$ and $\phi 2$.

1. Conditional Jump

Refer to Table I and assume that the address now being accessed is number three. The inputs to D1 and D2 are 1 and 0 respectively (see "ADR FIELD") ; line two (010) has been selected (see "SEL FIELD") and the operation coded as 000 is being executed by some hardware external to the model (see "OPCODE FIELD"). Note that the outputs of D1, D2 and D3 must currently be 011 respectively, since we assumed ROM word three was being accessed.

Eventually the clock ($\phi1$) will rise and the output of D1D2 will be 10, which implies that the address to be accessed is either 4 (100) or 5 (101) depending upon the output of D3. The output of D3 is the value of its input immediately prior to the $\phi1$ pulse, and this is the value of input (to MX) line two (010); thus it cannot be said which ROM word will be accessed next without specifying the earlier state of this signal. The effect of this example can be described by the ALGOL-like statement:

"IF INPUT(2) GO TO 5 ELSE GO TO 4"

where input(2) is treated as a logical variable.

Soon after $\phi1$, the outputs of the ROM start to change. Since it is not guaranteed that only one change in state will take place, $\phi2$ is kept high at this point, thus preventing the output of DMX from being affected by this spurious input.

One microsecond later, $\phi2$ goes low; consequently, the input of D3 is now defined and the right command is being enabled by one of the output lines of DMX.

2. Unconditional Jump

In the example described above, if it was known that input line two (010) had the value zero (it could be physically connected to ground), then the next address would have been forced to four. On the other hand, if it had the value 1 (connected to the power supply), an unconditional jump to location five would have resulted.

Therefore, to implement the unconditional jump, it suffices to save two input lines to MX and set them to 1 and 0 respectively.

3. Execution of a Predefined Process

It can be seen from the two previous examples that the output of DMX depends upon the particular address being accessed. By proper selection of the contents of the "next address" field, it is therefore possible to make the ROM

cause the execution of sequences of processes, as will be described.

Assuming that this hardware was to be used as control unit for an Arithmetic and Logic Unit of a computer, certain basic functions would be needed, such as adder, multiplier, divider, comparator, etc. These basic functions are collectively called "microspec functions" by Husson (Ref. 2). The microspec function has one enable line that activates it.

The hardware in the example allowed coding of eight possible operations. Therefore, if the output lines of DMX were connected to suitable microspec functions, up to eight different predefined processes could be selected and executed.

V. ALMIC - AN ASSEMBLY - LEVEL LANGUAGE FOR MICROPROGRAMMING

A. MOTIVATION

Given the basic hardware model described in chapter IV, the next task was the actual programming of the ROM's to generate the control sequences required by the /360 channel. This meant:

1) find the bit patterns to be stored in each field of each address;

2) put them on paper;

3) actually write them into the ROM.

The last operation was relatively easy, because all that is required is equipment already available. However, the first two proved not only tedious but also highly error-prone. In the case under study it was estimated that a 256 word, 16 bits per word, store would be needed, which implied a sizable number of bit strings to be input via a teletypewriter. In case an error was detected, or a change sought, most of the work would have to be done again.

It was decided that a higher level language would be desirable to allow straightforward description of control sequences and to automate their translation into ROM bit patterns. This required the design of a software package to support it and, due to time constraints, it was agreed that an assembler-level language would be more reasonable and still helpful.

The general format of a statement in the assembler language is given by the example:

34 : 28 , ADROUT,STAIN.

where the number before the colon (34) is the address where the statement is to be stored; the first field (28) is the next address (not the "next basic address" mentioned in chapter IV; the assembler will take care of this detail); ADROUT, in the example, stands for "select the decision line ADROUT" and the third field is the operation to be performed, "raise line StaIn (Status In)" in this case.

It is to be understood by this example that the next instruction will be in the address given by:

{28 + (current value of ADROUT, 1 or 0)}

therefore 28 or 29.

B. THE SOFTWARE PACKAGE

1. Introduction

The model presented in chapter IV was intended to be used in any sequential microprogrammed circuit. Therefore, before attempting to write programs for any specific hardware configuration, it is necessary to furnish the assembler with the following information:

1) number of addresses in the ROM;

2) number of bits in each field of a ROM word;

3) list of mnemonics used to represent the input lines to MX;

4) list of mnemonics used to represent the opcodes (or microspec functions).

2. __Functional Description and Use of the Software Package__

The package is composed of three main programs:

a) the DATA GENERATOR

b) the TABLE GENERATOR

c) the ASSEMBLER

In addition there are 13 subroutines: INIT, GNC, CONV, GET, PUT, ICON, PAD, ERROR, WRITEL, FORM, CCNOUT, SCAN, PUNCH.

a. The Data Generator

(1) __Purpose__. Generate input data for the Table Generator.

(2) __Input__. Input is in free-format 80-column records, with different elements separated by commas, except where otherwise noted. Blanks are always irrelevant, therefore "2 5 6 , 34." is the same as "256,3 4.". The following data is required:

(a) one card with the number one in column one;

(b) the number of fields in a ROM word, followed by a comma. This is necessary since it is allowed to separate the opcode field into as many sub-fields as wanted, thus providing the capacity to execute several operations simultaneously;

(c) the number of bits in each field of the ROM word;

(d) list of mnemonics used to represent the input lines to MX. The last mnemonic is to be followed by a period, not a comma;

(e) list of mnemonics used to represent the microspec functions. The last mnemonic (in each sub-field, if more than one is used) is to be followed by a period, not a comma.

(3) Output. The output is in form of punched cards ready to be fed to the Table Generator.

b. The Table Generator

(1) Purpose. Sets up tables to be used by the Assembler.

(2) Input. Input is in free-format 80-column records, with different elements separated by commas, except where otherwise noted. Blanks are always irrelevant. The following data is required :

(a) one card with the number two in column one;

(b) number of fields in each RCM word, followed by a comma;

(c) number of bits in each field, followed by a comma;

(d) list of mnemonics used to represent the input lines to MX. Each mnemonic is to be followed (after a comma) by its corresponding binary code;

(e) list of mnemonics used to represent the microspec functions, each mnemonic being followed (after a comma) by its corresponding binary code.

(3) Output. Fortran DATA statements ready to be inserted into the "Block Data" subprogram for use with the Assembler.

c. The Assembler

(1) Purpose. Converts statements of the form:

<label> : <address>,<select line>,<opcode>.

for example: 25 : 36 , SELOUT, DPSELOUT.

into bit patterns suitable to program a ROM.

(2) Input. The first card must have the number three in column one. For the program itself, input is in free-format 80-column records. Comments can be interspersed with (and even within) statements, provided

they are enclosed between the signs "<" and ">" . The card after the last in the program being assembled must have a "*" in column one.

(3) Output. Paper tape in a format suitable to program a ROM.

VI. IMPLEMENTATION OF A MICROPROGRAMMED INTERFACE

This section is composed of two parts; part A contains a description of the procedure used to implement the interface. In part B an example is given to illustrate and clarify the procedure described in part A.

A. OVERVIEW

The following steps should be adopted in designing a microprogrammed device using the hardware and software presented in chapters IV and V :

Step 1. Make a flowchart representation of the behavior of the device. This flowchart is to use the "binary decision" and the "predefined process" boxes only.

Step 2. Count the number of distinct decision variables. Call it $\underline{m}$.

Step 3. Count the number of distinct predefined processes. Call it $\underline{n}$.

Step 4. Count the number of decision boxes. Call it $\underline{p}$.

Step 5. Determine the number of fields (not bits) to be used in microprogramming the ROM. The least number is three, and will be greater if and only if more than one microspec function has to be activated at the same time.

Step 6. Determine the number of bits in each field. For the "next basic address" field it will be:

$$[\log_2 2p] - 1$$

where [x] means the least integer not less than x. For the "select field" the number of bits will be:

$$a = [\log_2 m]$$

For the "opcode field" it will be $[\log_2 n]$.

Step 7. Choose the component to play the role of MX. It will be a Data Selector/Multiplexer with at least "a" input bits.

Step 8. Choose the component to play the role of DMX. It will be a Decoder/Demultiplexer of capacity at least n to n^2.

Step 9. Design the hardware necessary to implement the microspec functions according to the specific needs of the project.

Step 10. Run the Data Generator using as inputs:

a) number of fields in each ROM word, followed by a comma;

b) number of bits in each field of the ROM, each followed by a comma;

c) list of mnemonics used to represent the input lines to MX. Each mnemonic is to be followed by a comma, except the last one, which shall be followed by a period;

d) list of mnemonics used to represent the microspec functions. Each mnemonic is to be followed by a comma, except the last one, which shall be followed by a period.

Step 11. Run the Table Generator using the output of the Data Generator as its input.

Step 12. Insert the output of the Table Generator in proper place within the "Block Data" subprogram for use with the Assembler.

Step 13. Using the algorithm presented in Appendix B, label the boxes of the flowchart.

Step 14. Using the algorithm presented in Appendix C, write the microprogram and punch it.

Step 15. Run the Assembler using the microprogram as input. The program is currently written in FORTRAN for a XDS-9300 computer. To run the Assembler in other computers minor changes are necessary. As examples, the compiler may not accept more than 20 continuation cards which requires breaking up the "DATA MEMORY" statement inside the "Block Data" subprogram into smaller statements; the logical number for the output unit (paper tape punch) was assumed to be seven.

The output of the Assembler is a paper tape ready to be fed to the MCS-8 PROM Programming System.

B. EXAMPLE

The I/O Interface for the System/360 will be used to demonstrate the method just described. Figures 4 and 5 contain a block diagram of the complete circuit.

From Figure 4 it can be seen that inputs to MX number 0 and 1 were reserved to implement unconditional jumps. Inputs two thru six are outbound tags from the channel. Input seven will be provided by the associated microcomputer, having the value of one whenever the microcomputer, or the device attached to it, is busy. Inputs nine and ten are provided by the hardware shown in Figure 5. Input ten is tapped from the Status In line.

Figure 5 displays the executive part of the interface hardware. Output line number zero for the DMX was reserved to represent "no operation" to be performed. Lines one and two respectively raise and drop the "channel-initiated-sequence" line which is fed to MX in Figure 4. The squares with the letters R and D are latches whose outputs switch to 1 when R (raise) is zero and to zero when D (drop) is zero.

Output lines three and four implement the SelOut control. Lines five thru 13 control the multiplexing of

data, status and address into BusIn. At the same time, lines six and seven, nine and ten and 12 and 13 implement SerIn, AdrIn and StaIn respectively.

Whenever the microcomputer wants to send/receive information to/from the channel, it will raise ReqIn, which will be dropped by output line 16.

In the sample design which follows the "reset" and the "disconnect" sequences (described respectively under "Operational Out" and "Address Out" in chapter II) were not considered. The action to be taken in case of wrong parity on the address byte was also omitted.

Step 1. The flowchart will be as shown in Appendix A.

Step 2. The decision variables are: 0, 1, ADROUT, SELOUT, SUPOUT, COMOUT, SEROUT, CUBUSY, CHSEQ, OURADR, STAIN; therefore m = 11.

Step 3. The predefined processes are :NO OP, CHSEQ, DCHSEQ, PSELOUT, DPSELOUT, DATABUSIN, DSERIN, SERIN, STABUSIN, DSTAIN, STAIN, ADRBUSIN, DADRIN, ADRIN, CPLIN, DCPLIN, DREQIN, TSTADR. Therefore n = 18.

Step 4. There are 20 decision boxes, thus p = 20.

Step 5. Three fields only will be used, as there is no need for simultaneous execution of microspec functions.

Step 6. Number of bits in "next basic address field":

$$[\log_2 2 \times 20] - 1 = 5$$

number of bits in "select field": $a = [\log_2 11] = 4$

number of bits in "opcode field": $[\log_2 18] = 5$

The size of ROM address space will be the number of possible "next basic addresses", $2^5 = 32$, doubled (for the two different states of the address bit from D6, Figure 4); a total of 64 words in this case. Each word shall have at least 14 bits. Intel's 1702A has 256 words, eight bits per word, and is reprogrammable. Connecting two of them as in figure 6 a 256 word, 16 bits per word, store is obtained.

Step 7. MX will have four inputs; Signetics N74150 is suitable.

Step 8. DMX has 32 outputs; since no decoder is available with so many outputs, two Signetics N74154's will be used, connected as in Figure 7. Bit 0 of ROM will act as "chip selector".

Step 9. There are eight microspec functions of the form: "Raise/Drop line...", namely, CHSEQ/DCHSEQ, PSELOUT/DPSELOUT, STAIN/DSTAIN, SERIN/DSERIN, ADRIN/DADRIN, OPLIN/DOPLIN, DREQIN, TSTADR.

The logic circuit to perform this operation will have two inputs (Raise and Drop, or R and D) and one output. The inputs should be level-triggered by the low signal, as this is the output available from DMX. Therefore, the corresponding truth table is as depicted in Figure 8a; Figure 8b shows one possible implementation.

For the three functions which deal with BusIn (DATABUSIN, STABUSIN, ADRBUSIN) a set of eight AND-OR gates working as a multiplexer will suffice. The data and status bytes will be provided by the microcomputer, while the address byte will come directly from BusOut.

The address of an I/O device can be any eight-bit pattern. The address checking function (TSTADR) will have eight inputs, to be fed by BusOut. It is necessary to have some switching capability in order to select, at installation time, the range for valid addresses. The output is one line (OURADR), which will have the value one whenever the input address is within range. Figure 9 shows the logical circuit to perform the function. The switch S will be in position one for those bits which must be one for the address to be accepted, in position two for those bits which must be zero, and in position three for those bits which are irrelevant.

Step 10. The input to the Data Generator is displayed in Figure 10a, whereas part b of the same figure shows the output obtained.

Step 11. The output of the Table Generator is displayed in Figure 11.

Step 12. The output of the Table Generator is inserted in the "Block Data" subprogram.

Step 13. The flowchart of Appendix A was numbered using the algorithm described in the previous section.

Step 14. The resulting microprogram is listed in Figure 12.

Step 15. Using the input shown in Figure 12 to run the Assembler, the output will be a paper tape ready to microprogram the ROM.

VII. CONCLUSION

This thesis dealt with the design of a microprogrammed I/O interface to be used in a communications network at the Naval Postgraduate School.

A basic hardware approach suitable to most microprogrammed sequential applications was described along with an assembler-level language for microprogramming.

The fact that it was possible to devise an algorithm to write the AIMIC microprogram suggests that it might be feasible to improve the software package to the point where the flowchart itself, and not the program, would be used as input to the system; the flowchart, as used here, can be represented by some sort of binary tree.

In order to implement and test the interface it is necessary to incorporate in this design the hardware and also the microinstructions needed to handle the exchange of information between the device and the microcomputer.

APPENDIX A

This appendix contains the flowchart used to implement the I/O interface between the System/360 channel and the device described in this thesis. It was obtained from Appendix C of Ref. 1 by eliminating all boxes "under responsibility of the channel" and by adding others necessary to specify operations to be performed by the device.

As to the mnemonics used, the following general rules apply:

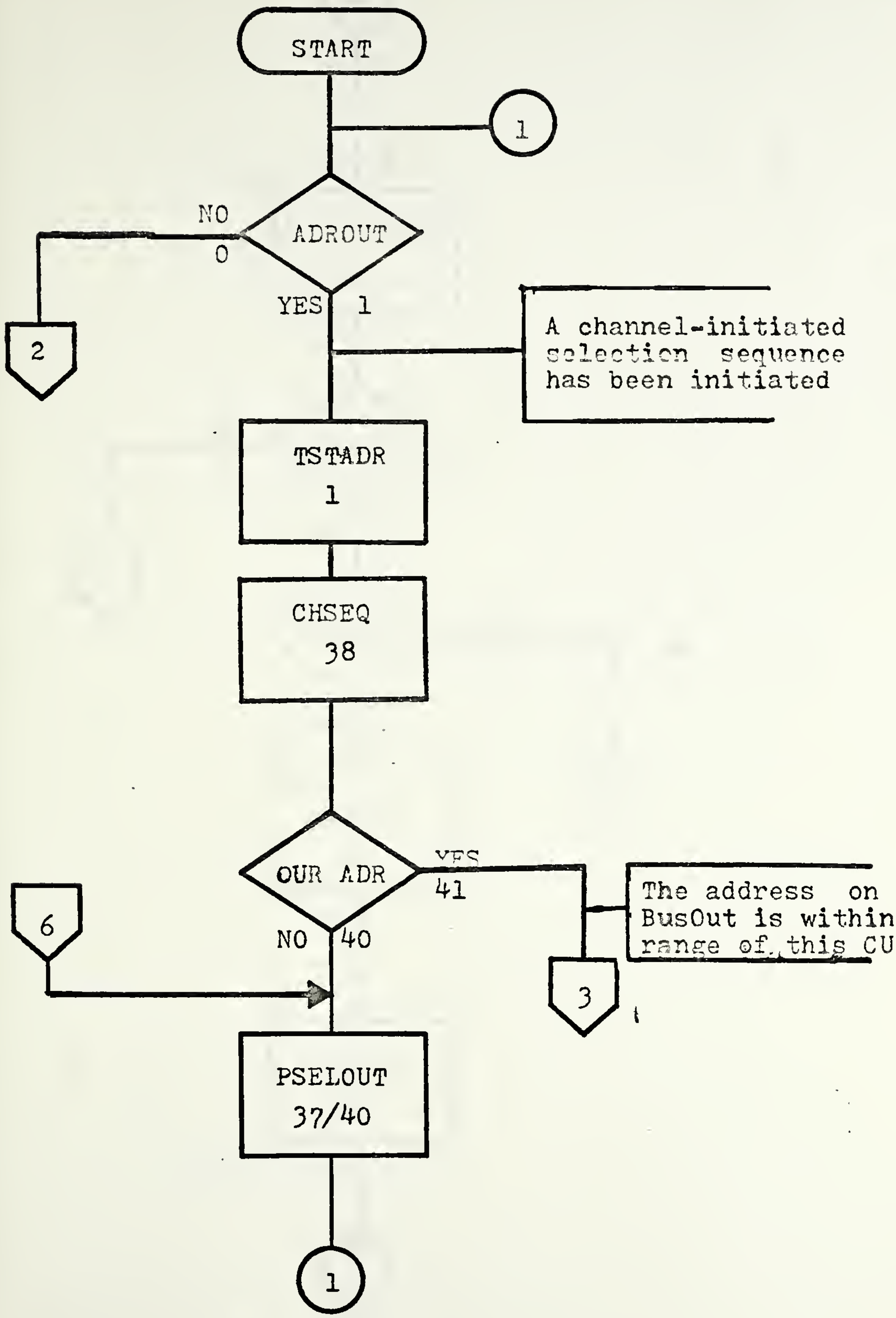
a) the name of a line inside a decision box means: "Is the line up?";

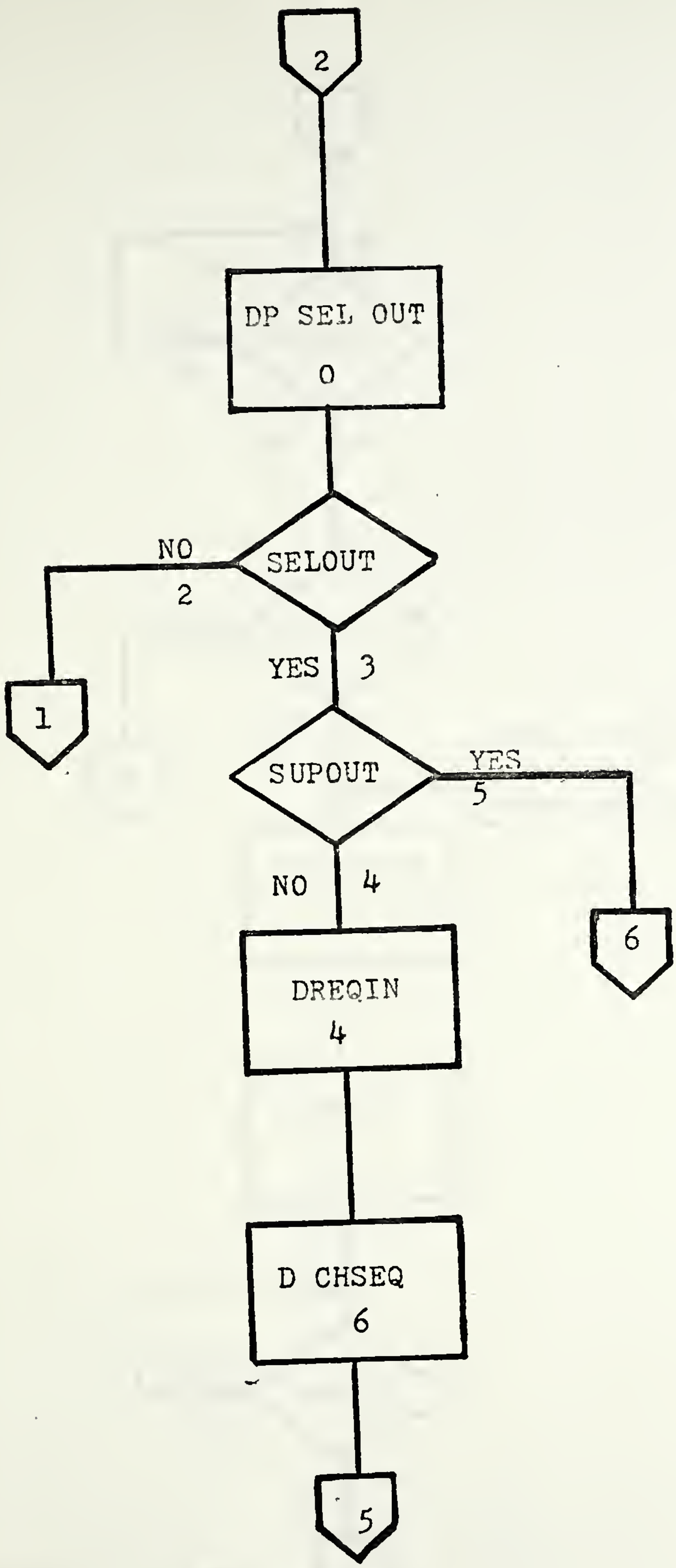
b) the name of a line inside a process box means: "Raise line";

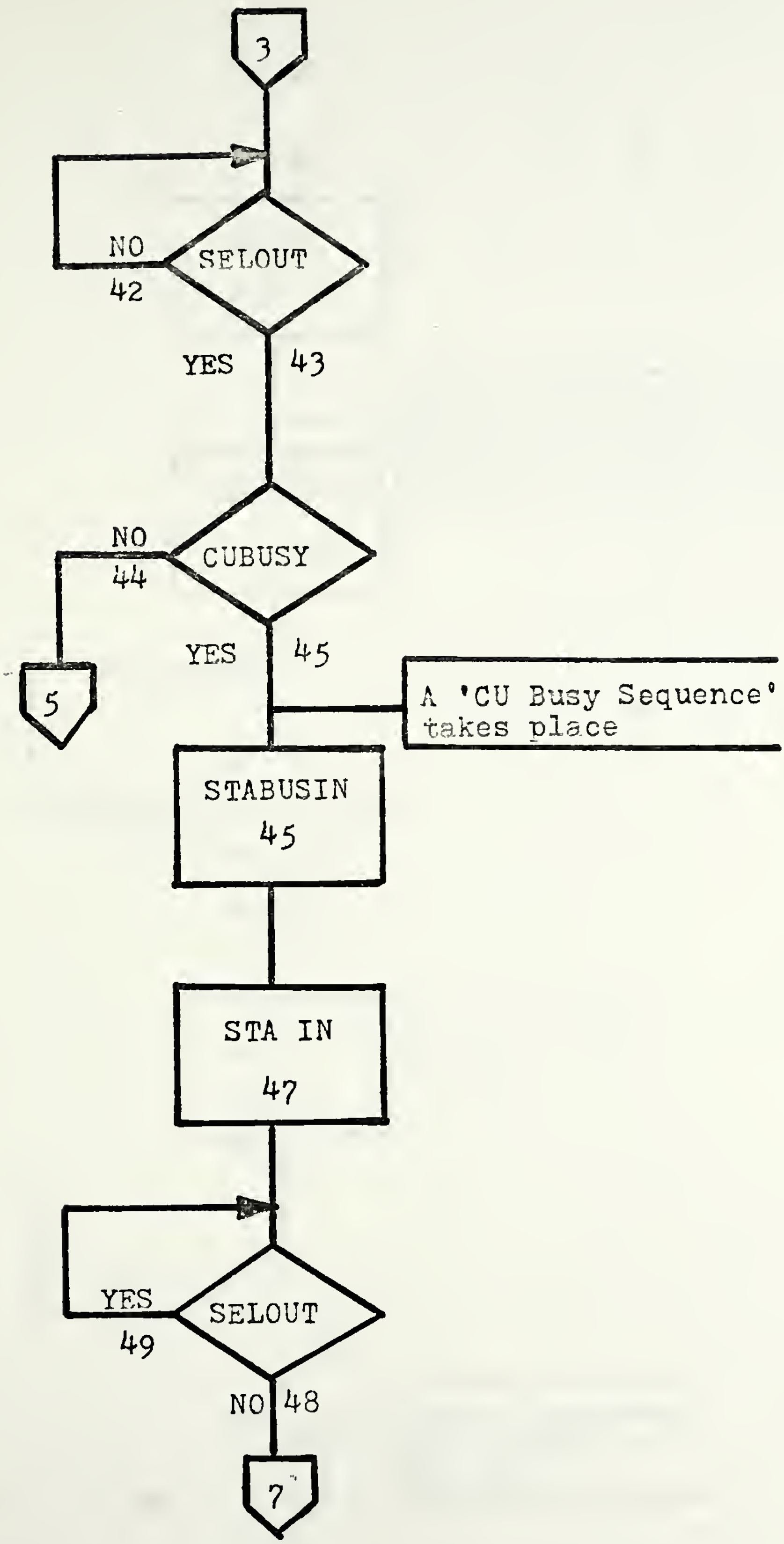
c) the name of a line inside a process box when preceded by the letter "D" means "Drop line".

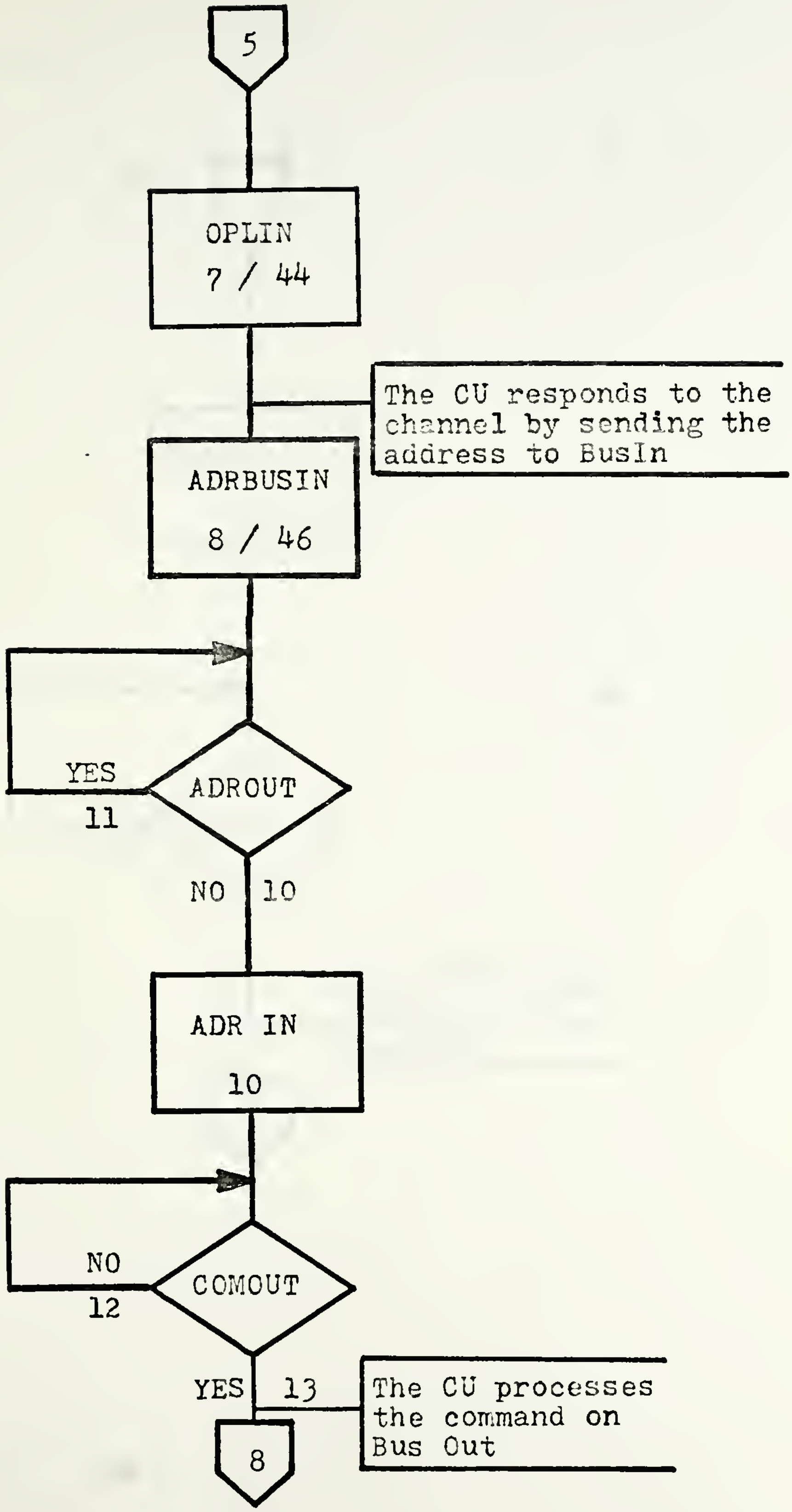
The lines are :

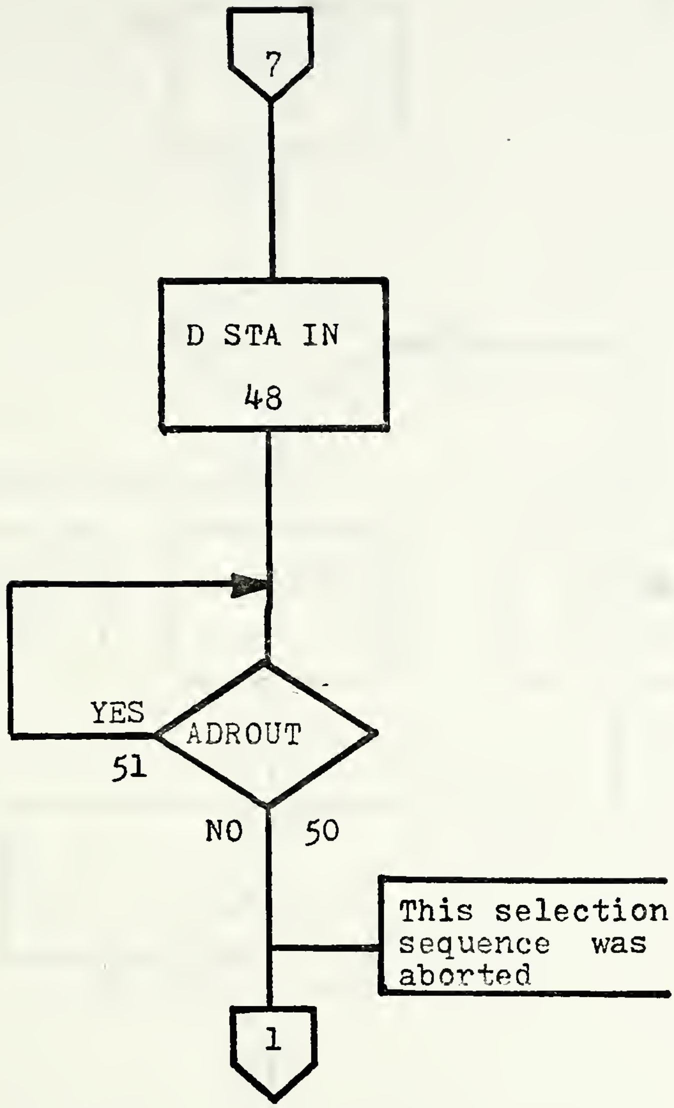
ADRBUSIN-Address byte to BusIn
ADRIN-Address In
ADROUT - Address Out
CHSEQ- Channel-Initiated-Sequence
COMOUT - Command Out
DATABUSIN-Data byte to Bus In
OPLIN-Operational In
PSELOUT - Propagate Select Out
REQIN-Request In
SELOUT -Select Out
SERIN-Service In
SEROUT - Service Out
STABUSIN-Status byte to BusIn
STAIN-Status In
SUPOUT - Suppress Out

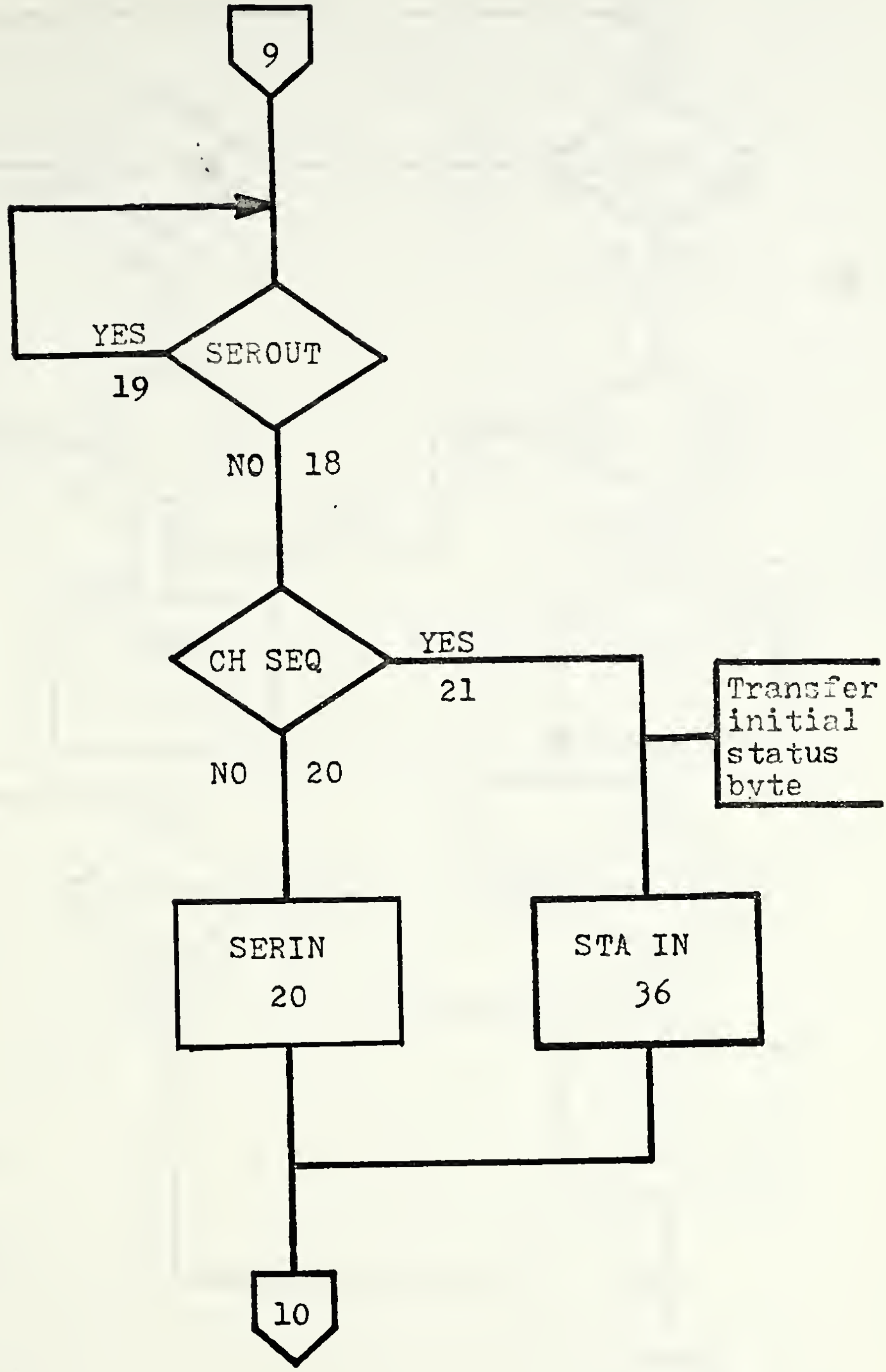

START
1
NO
0
ADROUT
YES
1
2
A channel-initiated selection sequence has been initiated
TSTADR
1
CHSEQ
38
OUR ADR
YES
41
The address on BusOut is within range of this CU
6
NO
40
3
PSELOUT
37/40
1

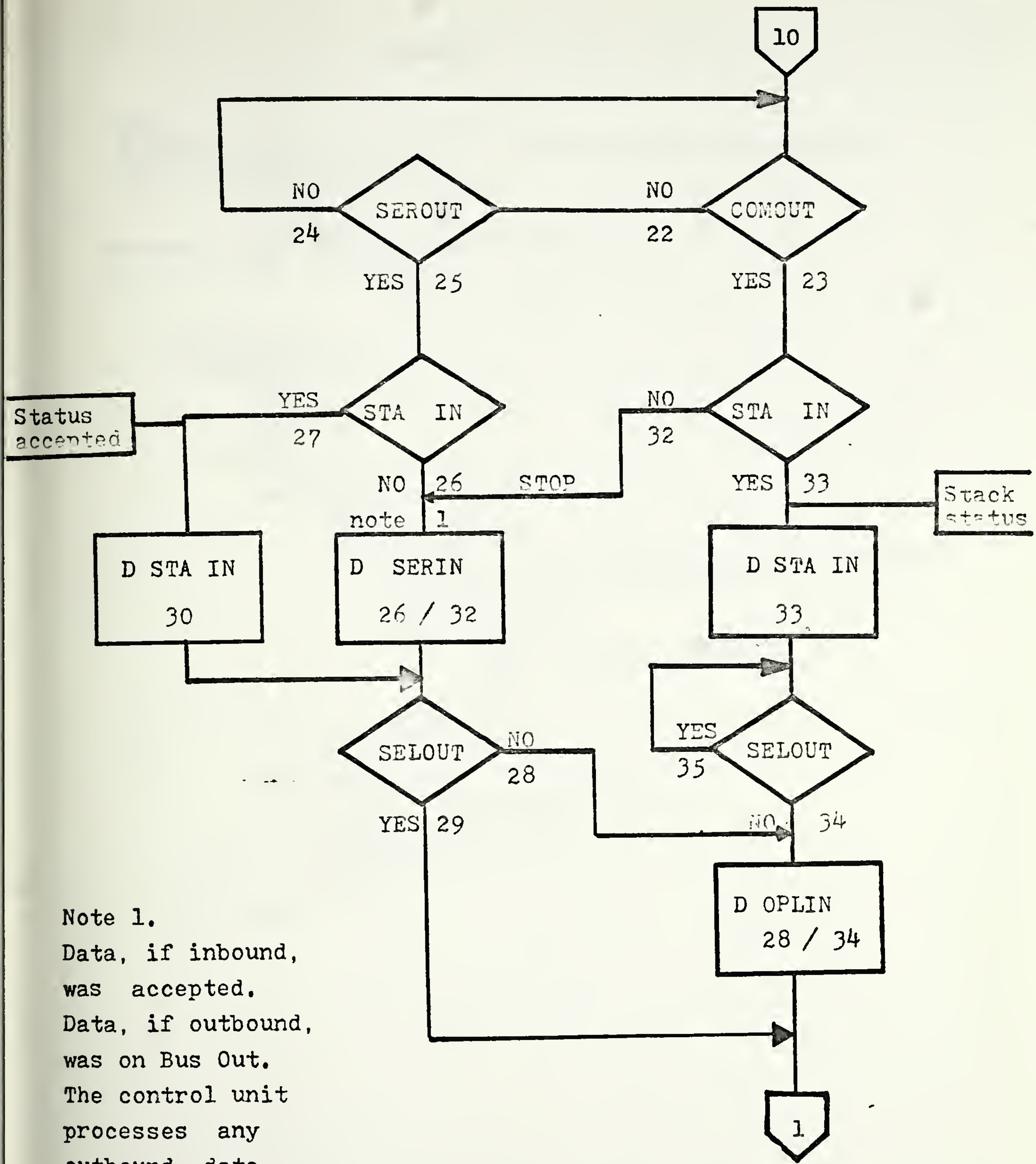

2
DP SEL OUT
0
NO
2
SELOUT
YES
3
1
SUPOUT
YES
5
NO
4
6
DREQIN
4
D CHSEQ
6
5


3
NO
SELOUT
42
YES
43
NO
CUBUSY
44
YES
45
5
A 'CU Busy Sequence' takes place
STABUSIN
45
STA IN
47
YES
SELOUT
49
NO
48
7


5
OPLIN
7 / 44
The CU responds to the channel by sending the address to BusIn
ADRBUSIN
8 / 46
YES
11
ADROUT
NO
10
ADR IN
10
NO
12
COMOUT
YES
13
The CU processes the command on Bus Out
8


7
D STA IN
48
YES
51
ADROUT
NO
50
This selection sequence was aborted
1


9
YES
SEROUT
19
NO
18
CH SEQ
YES
21
Transfer initial status byte
NO
20
SERIN
20
STA IN
36
10



Note 1.
Data, if inbound, was accepted.
Data, if outbound, was on Bus Out.
The control unit processes any outbound data.

APPENDIX B

```
BEGIN
INTEGER I ;
RECORD POINTER(INTEGER LAST; REFERENCE(POINTER)NEXT);
REFERENCE(POINTER) TOP;


COMMENT : SET YOURSELF AT 'START' BOX ;
I := 0 ; TOP := NULL ;


A : TAKE NEXT BOX ;
    IF RECTANGULAR
    THEN BEGIN
         LABEL IT WITH  I ;
         I := I + 1 ;
         GO TO A
         END
    ELSE BEGIN
         IF ALREADY VISITED
         THEN BEGIN
              IF TOP = NULL
              THEN GO TO STOP
              ELSE BEGIN
                   COMMENT : SET YOURSELF AT 'YES' BRANCH
                   CORRESPONDING TO LAST(TOP);
                   TAKE NEXT BOX ;
                   IF RECTANGULAR
                   THEN BEGIN
                        LABEL IT WITH LAST(TOP) ;
                        TOP := NEXT(TOP) ;
                        GO TO A
                        END
                   ELSE GO TO C
                   END
              END
         ELSE BEGIN
              IF I IS ODD THEN I := I + 1 ;
              LABEL 'NO' BRANCH WITH I ;
              LABEL 'YES' BRANCH WITH I + 1 ;
              I := I + 2 ;
              GO TO B
              END
         END;
```

```
B : TOP := POINTER (LABEL OF 'YES' BRANCH, TOP) ;
    TAKE BOX CONNECTED TO 'NO' BRANCH ;
    IF RECTANGULAR
    THEN BEGIN
         LABEL IT WITH I - 2 ;
         GO TO A
         END
    ELSE BEGIN
         IF ALREADY VISITED
         THEN BEGIN
              IF TOP = NULL
              THEN GO TO STOP
              ELSE BEGIN
                   COMMENT : SET YOURSELF AT 'YES'
                   BRANCH CORRESPONDING TO LAST(TOP) ;
                   TAKE NEXT BOX;
                   IF RECTANGULAR
                   THEN BEGIN
                        LABEL IT WITH LAST(TOP) ;
                        TOP := NEXT(TOP) ;
                        GO TO A
                        END
                   ELSE GO TO C
                   END
              END
         ELSE BEGIN
              IF I IS ODD THEN I := I + 1 ;
              LABEL 'NO' BRANCH WITH I ;
              LABEL 'YES' BRANCH WITH I + 1 ;
              I := I + 2 ;
              GO TO B
              END
         END ;


C : TOP := NEXT(TOP) ;
    IF ALREADY VISITED
    THEN BEGIN
         COMMENT : SET YOURSELF AT 'YES'
         BRANCH CORRESPONDING TO LAST(TOP) ;
         TOP := NEXT(TOP);
         GO TO A
         END;
    IF I IS ODD THEN I := I + 1 ;
    LABEL 'NO' BRANCH WITH I ;
    LABEL 'YES' BRANCH WITH I + 1 ;
    I := I + 2 ;
    GO TO B ;
STOP : END.
```

APPENDIX C

In order to write an ALMIC statement, all that is needed is to write the address number followed by a colon and then:

a) for the "next address" field:

1) find the label in the flowchart corresponding to the desired address;

2) the next address is the label of the next box if it is a process box or the label of the "no" branch otherwise.

b) for the "select" field :

1) find the label in the flowchart corresponding to the desired address;

2) if the next box is a decision box, use its contents as "select" field;

3) if the next box is an even numbered process box, use zero; otherwise use 1 for "select" field.

c) for the "opcode" field :

1) find the label in the flowchart corresponding to the desired address.

2) if it belongs to a process box use its contents as "opcode" ; otherwise leave blank.

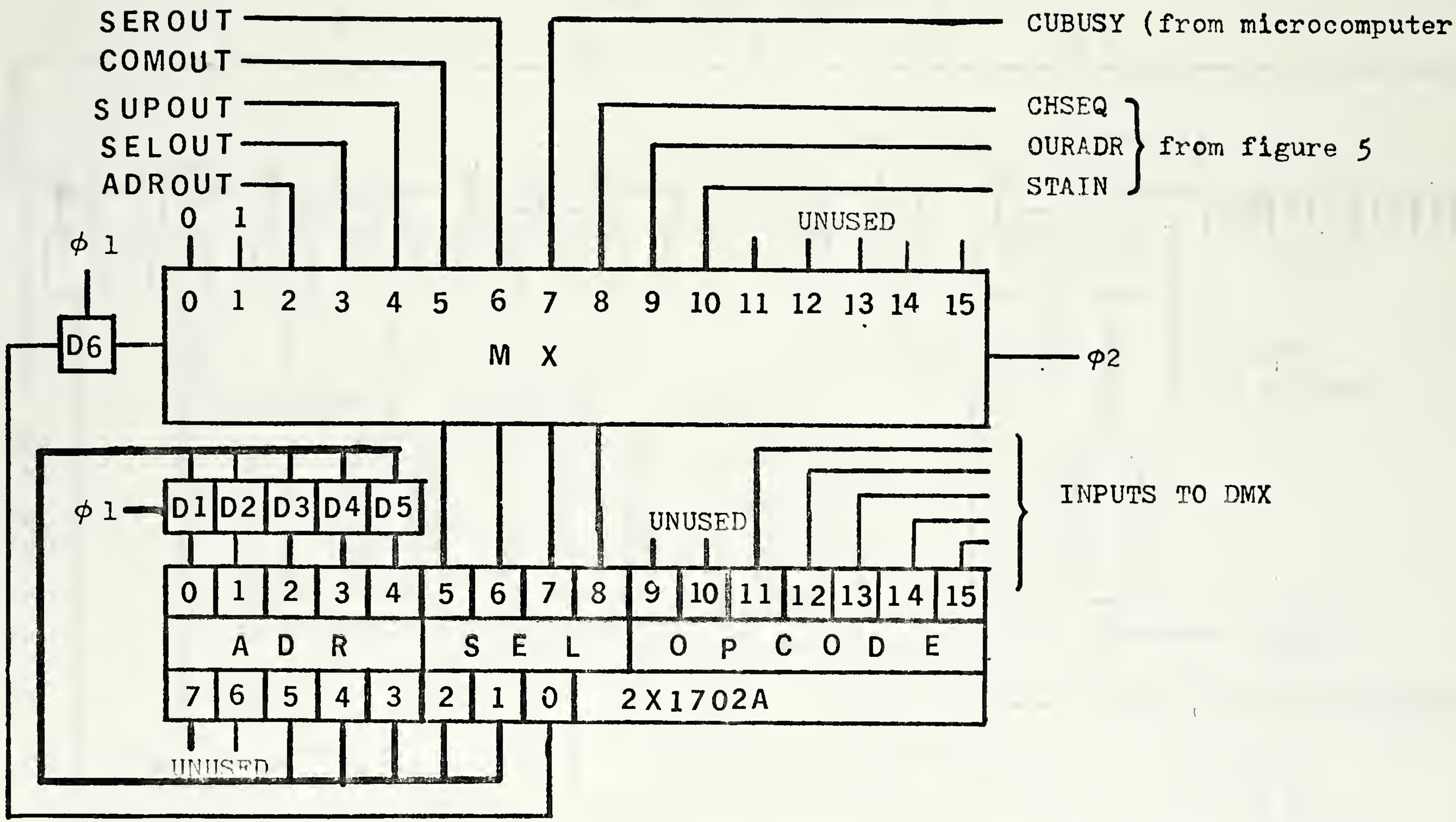


Figure 4. Control portion of the interface

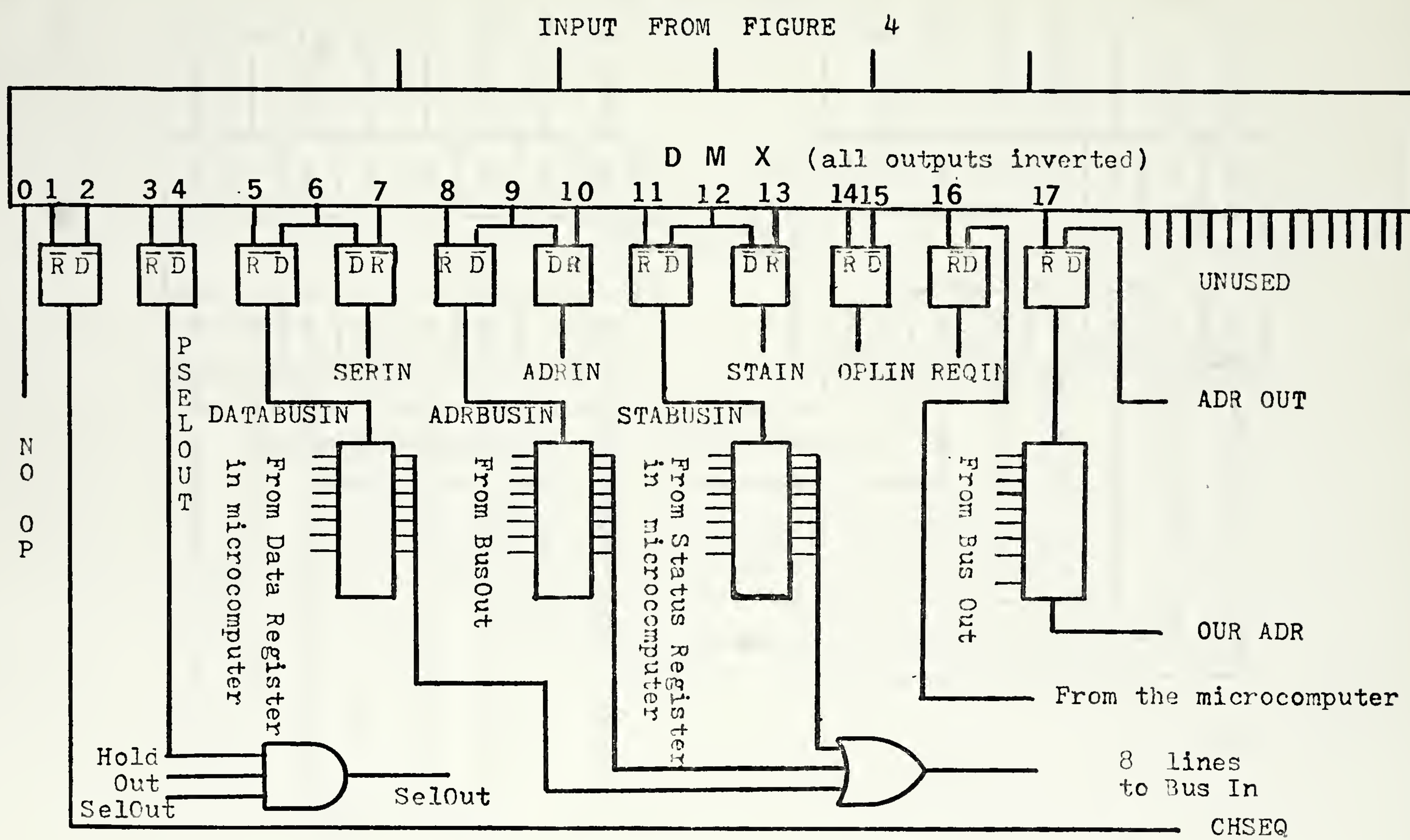


Figure 5. Executive part of the interface

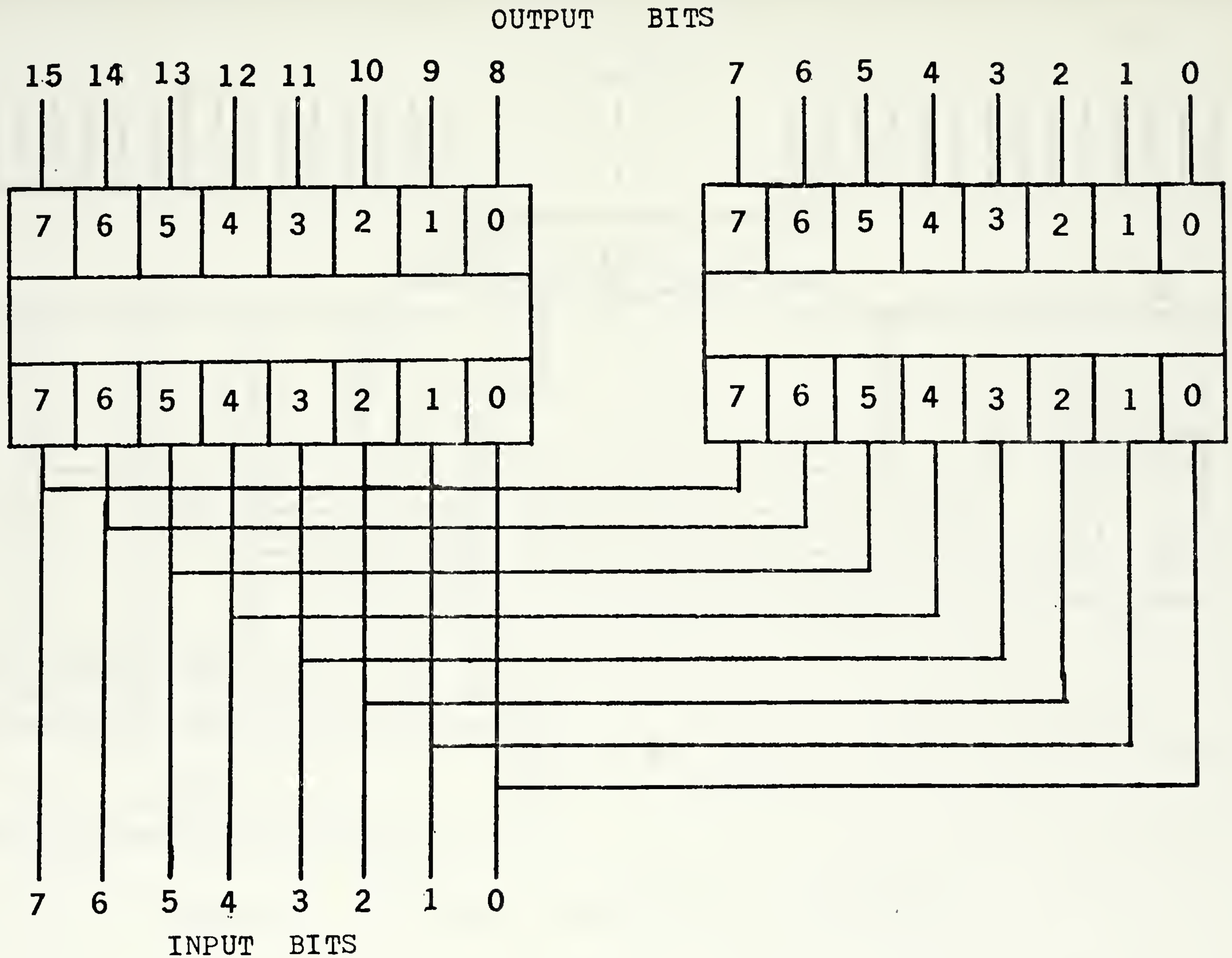


Figure 6. Parallel connection of two ROMs

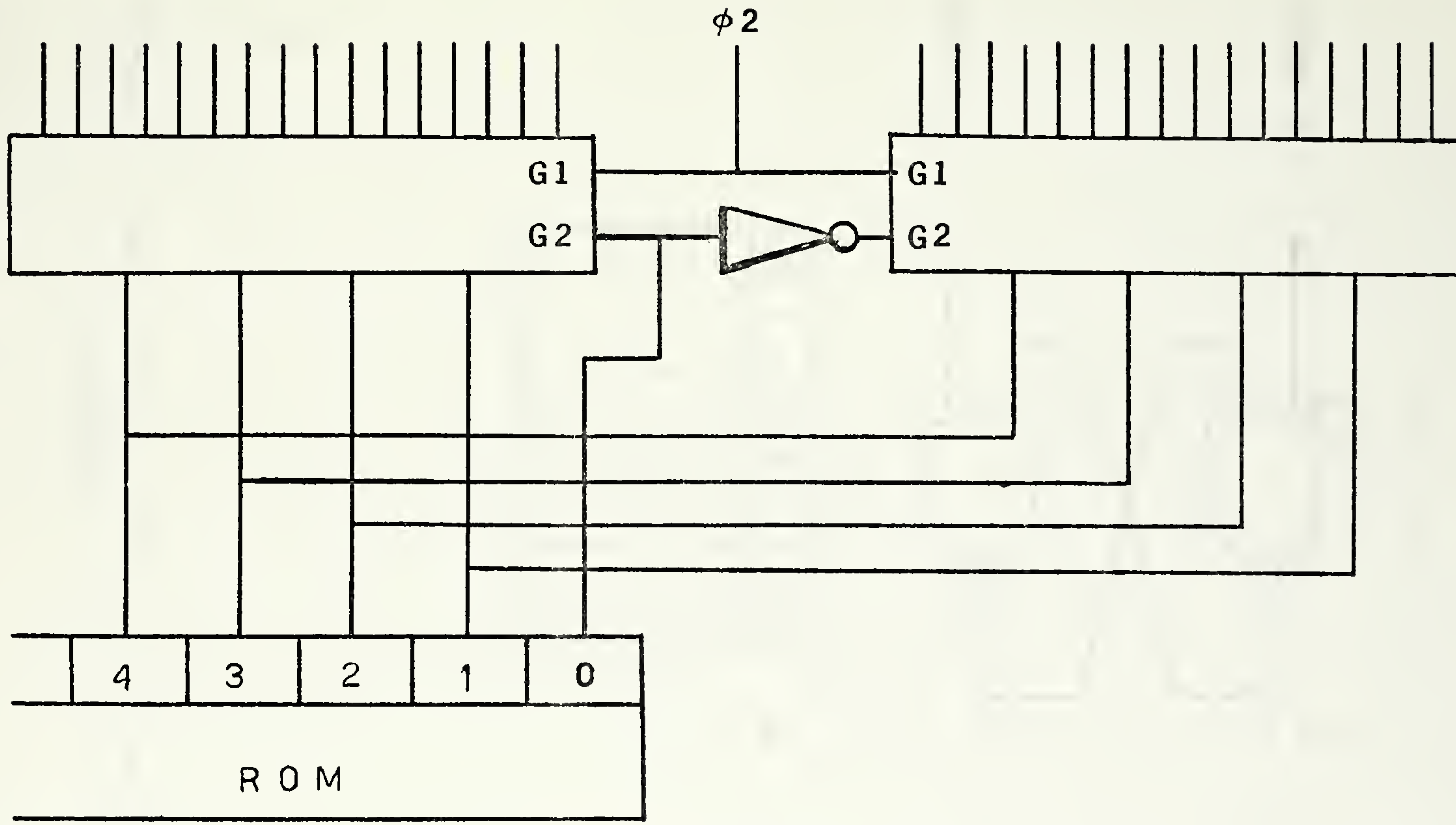
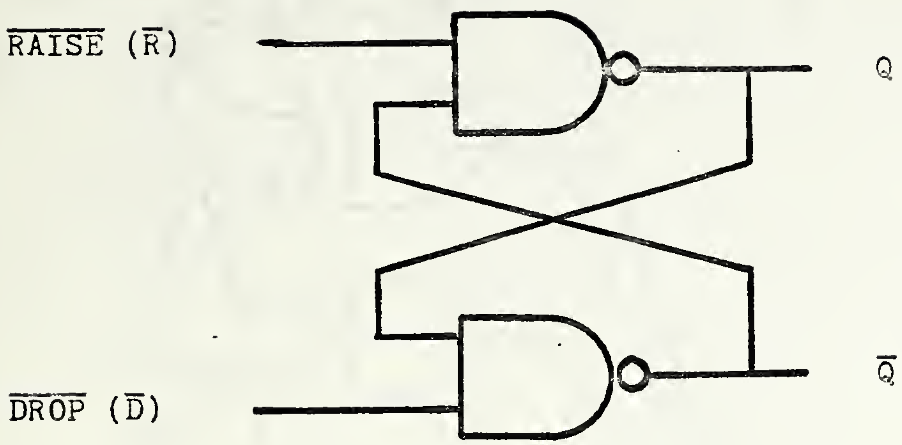


Figure 7. Parallel connection of Decoders



(b)

$\overline{R}$	$\overline{D}$	Q	$\overline{Q}$	Q^+	$\overline{Q}^+$
0	0	X	X	X	X
0	1	X	X	1	0
1	0	X	X	0	1
1	1	X	X	Q	$\overline{Q}$

(a)

Figure 8. Latch circuit for the Raise/Drop Line function

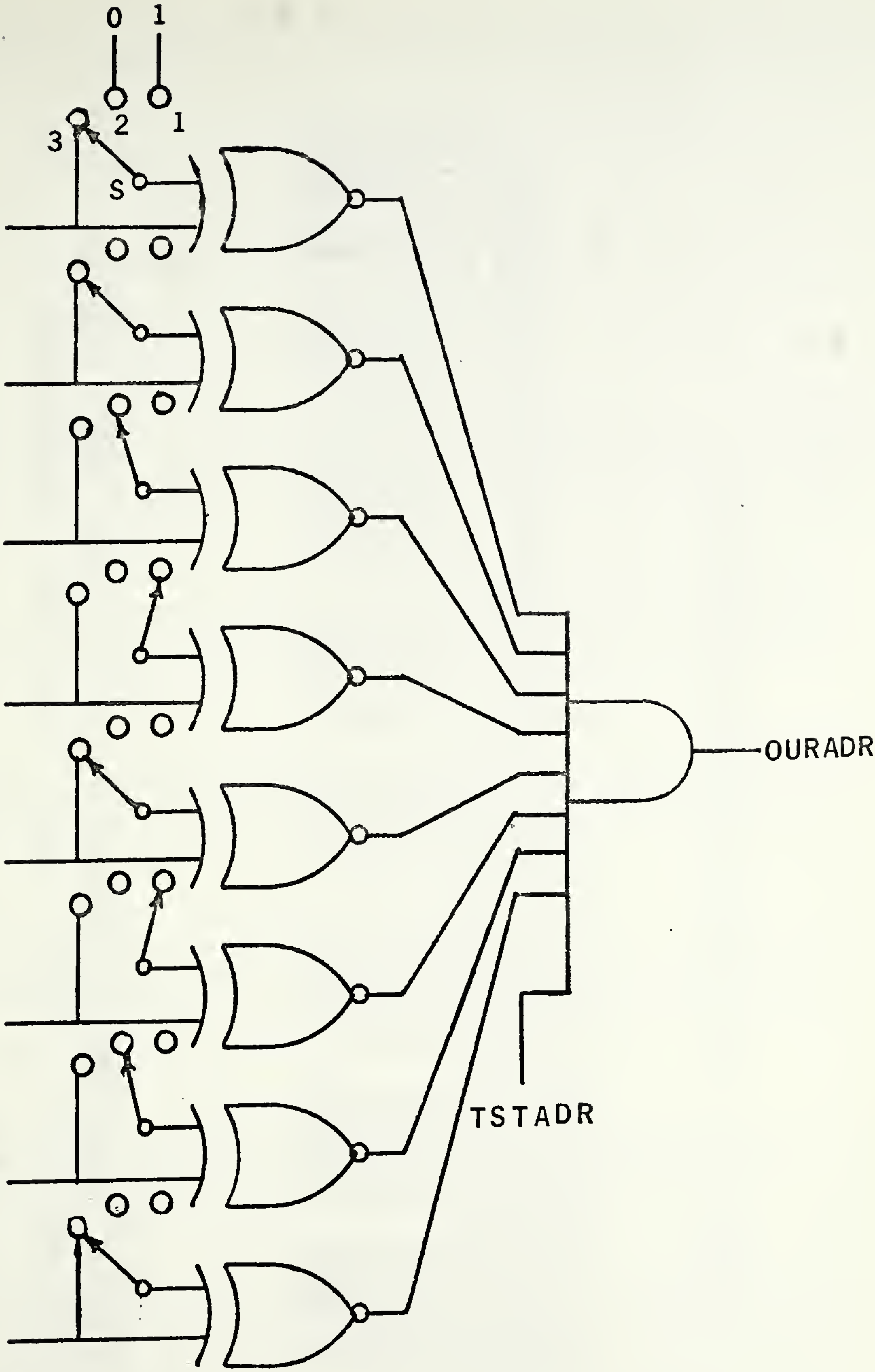


Figure 9. Address-checking function

```
1
3, 5, 4, 7,
  0, 1, ADROUT, SELOUT, SUPOUT, COMOUT, SEROUT,CUBUSY, CHSEQ, OURADR,
  STAIN.
 0, CHSEQ, DCHSEQ, PSELOUT, DPSELOUT, DATABUSIN, DSERIN, SERIN, ADRBUSIN,
 DADRIN, ADRIN, STABUSIN, DSTAIN, STAIN, OPLIN, DOPLIN, DREQIN, TSTADR.
```

(A)

```
2
 3, 5, 4, 7,
  0,00000,  1,00000,  2,00001,  3,00001,  4,00010,  5,00010,  6,00011,  7,00011,
  8,00100,  9,00100, 10,00101, 11,00101, 12,00110, 13,00110, 14,00111, 15,00111,
 16,01000, 17,01000, 18,01001, 19,01001, 20,01010, 21,01010, 22,01011, 23,01011,
 24,01100, 25,01100, 26,01101, 27,01101, 28,01110, 29,01110, 30,01111, 31,01111,
 32,10000, 33,10000, 34,10001, 35,10001, 36,10010, 37,10010, 38,10011, 39,10011,
 40,10100, 41,10100, 42,10101, 43,10101, 44,10110, 45,10110, 46,10111, 47,10111,
 48,11000, 49,11000, 50,11001, 51,11001, 52,11010, 53,11010, 54,11011, 55,11011,
 56,11100, 57,11100, 58,11101, 59,11101, 60,11110, 61,11110, 62,11111, 63,11111.
0,0000,1,0001,ADROUT,0010,SELOUT,0011,SUPOUT,0100,COMOUT,0101,SEROUT,0110,CUBUSY
,0111,CHSEQ,1000,OURADR,1001,STAIN,1010.
0,0000000,CHSEQ,0000001,DCHSEQ,0000010,PSELOUT,0000011,DPSELOUT,0000100,DATABUSI
N,0000101,DSERIN,0000110,SERIN,0000111,ADRBUSIN,0001000,DADRIN,0001001,ADRIN,
0001010,STABUSIN,0001011,DSTAIN,0001100,STAIN,0001101,OPLIN,0001110,DOPLIN,
0001111,DREQIN,0010000,TSTADR,0010001.
```

(B)

FIGURE 10. INPUT TO/OUTPUT FROM THE DATA GENERATOR

```
      DATA NF /  3/,FWIDTH /  5,  4,  7/
      DATA ISTART/   1, 504, 614,04000/
      DATA MEMORY /         258,         514,         514,         514,
     1       514,        514,        514,        770,        514,
     1       514,        516,        514,        514,        770,
     1      1282,        514,        515,        518,        514,
     1       515,        514,       1794,        514,        770,
     1       520,        514,        515,        770,       2306,
     1       514,        771,        522,        514,        770,
     1       514,       2818,        515,        514,        771,
     1       514,        515,        515,        771,        770,
     1       515,        515,        771,       1026,        515,
     1       770,        771,       1282,        515,        770,
     1       771,       1538,        515,        771,        771,
     1      1794,        515,        771,        771,       2050,
     1       770,        514,        771,       2306,        770,
     1       514,        771,       2562,        770,        515,
     1       771,       2818,        770,        515,        772,
     1       514,        770,        770,        772,        770,
     1       770,        770,        772,       1026,        770,
     1       771,        772,       1282,        770,        771,
     1       772,       1538,        771,        514,        772,
     1      1794,        771,        514,        772,       2050,
     1       771,        515,        772,       2306,        771,
     1       515,        772,       2562,        771,        770,
     1       772,       2818,        771,        770,        773,
     1       514,        771,        771,        773,        770,
     1       771,        771,        773,       1027,        514,
     1       514,        773,       1283,        514,        514,
     1       773,       1539,        514,        515,        773,
     1      1795,        514,        515,        773,       2051,
     1       514,        770,        773,       2307,        514,
     1       770,        773,       2563,        514,        771,
     1       773,       2819,        514,        771,        774,
     1       515,        515,        514,        774,        771,
     1       515,        514,        774,       1027,        515,
     1       515,        774,       1283,        515,        515,
     1       774,       1539,        515,        770,        774,
     1      1795,        515,        770,        774,       2051,
     1       515,        771,        774,       2307,        515,
     1       771,        774,       2563,        770,        514,
     1       774,       2819,        770,        514,        775,
     1       515,        770,        515,        775,        771,
     1       770,        515,        775,       1027,        770,
     1       770,        775,       1283,        770,        770,
     1       775,       1539,        770,        771,        775,
     1      1795,        770,        771,        775,       2051,
     1       771,        514,        775,       2307,        771,
```

```
1         514,      775,     2563,      771,      515,
1         775,     2819,      771,      515,      776,
1         515,      771,      770,      776,      771,
1         771,      770,      776,     1027,      771,
1         771,      776,     1281,      514,      514,
1         514,      514,      514,      515,      514,
1         515,     1804,     3869,     6688,     7938,
1         515,      519,     7696,     5914,     8223,
1         514,      771,     1822,     8219,     6688,
1        7938,      770,      519,     3610,     6170,
1        8223,      515,      515,     1822,     4125,
1        6688,     7938,      771,      519,     3616,
1        3360,     7716,      515,      771,     1550,
1        4894,     4124,      770,      514,     1818,
1        8221,     3087,     7427,      514,      774,
1        7711,     3092,     6401,      514,      514,
1         514,      514,      514,      514,      514,
1         514,     1550,     4894,     4124,      514,
1         514,      514,      775,     3854,     4894,
1        4124,      514,      514,      515,      520,
1        6942,     4119,     6688,     7938,      514,
1         514,      771,     2319,     6942,     4119,
1        6688,     7938,      514,      515,      514,
1        2575,     3103,     3085,     8222,     5145,
1         514,      514,      770,      775,     3870,
1        4125,     5145,      514,      514,      771,
1         518,     7696,     7444,     6402,      514,
1         515,      771,     2316,     3869,     3360,
1        7700,     6402,      514,      770,      514,
1        1807,     3087,     7444,     6402,      514,
1         770,      515,     1548,     3869,     5145,
1         514,      515,      515,      521,     7711,
1        3085,     8222,     5145,      514,      515,
1         515,      775,     3370,     7948,     5145,
1         514,      515,      770,      518,     7711,
1        3092,     6402,      514,      771,      515,
1        1562,     6935,     5145,      514,      515,
1         771,      519,     3866,     6935,     5145,
1         514,      515,      771,      775,     3869,
1        4124,     5145,      514,      770,      514,
1         519,     7966,     7948,     3869,       0/
```

FIGURE 11. OUTPUT FROM THE TABLE GENERATOR

```
3
  :    ,
 0 :   2,   SELOUT    ,DPSELOUT  .
 1 :  38,    0        ,TSTADR    .
 2 :   0,   ADRCUT    ,          .
 3 :   4,   SUPCUT    ,          .
 4 :   6,    0        ,DREQIN    .
 5 :  37,    1        ,          .
 6 :   7,    1        ,DCHSEQ    .
 7 :   8,   0         ,OPLIN     .
 8 :  10,   ADROUT    ,ADRBUSIN  .
10 :  12,   COMOUT    ,ADRIN     .
12 :  12,   COMCUT    ,          .
13 :  14,   CHSEQ     ,DADRIN    .
14 :  16,   COMOUT    ,DATABUSIN .
15 :  16,   COMCUT    ,STABUSIN  .
16 :  18,   SERCUT    ,          .
17 :  16,   COMOUT    ,          .
18 :  20,   CHSEQ     ,          .
19 :  18,   SERCUT    ,          .
20 :  22,   COMOUT    ,SERIN     .
22 :  24,   SERCUT    ,          .
23 :  32,   STAIN     ,          .
24 :  22,   COMCUT    ,          .
25 :  26,   STAIN     ,          .
26 :  28,   SELOUT    ,DSERIN    .
27 :  30,    0        ,          .
28 :   0,   ADROUT    ,DOPLIN    .
29 :   0,   ADROUT,   ,          .
30 :  28,   SELCUT    ,DSTAIN    .
32 :  28,   SELCUT    ,DSERIN    .
33 :  34,   SELCUT    ,DSTAIN    .
34 :   0,   ADRCUT    ,DOPLIN    .
35 :  34,   SELCUT    ,          .
36 :  22,   COMCUT    ,          .
37 :   0,   ADRCUT    ,PSELOUT   .
40 :  0 ,   ADROUT    ,PSELOUT   .
41 :  42,   SELCUT    ,          .
42 :  42,   SELCUT    ,          .
43 :  44,   CUBUSY    ,          .
44 :  46,    0        ,OPLIN     .
45 :  47,    1        ,STABUSIN  .
46 :  10,   ADROUT    ,ADRBUSIN  .
47 :  48,   SELCUT    ,STAIN     .
48 :  50,   ADRCUT    ,DSTAIN    .
49 :  48,   SELOUT    ,          .
50 :   0,   ADRCUT    ,          .
51 :  50,   ADRCUT    ,          .
*
```

FIGURE 12. THE MICROPROGRAM FOR THE INTERFACE

```
C       FOR EASE OF CHARACTER  MANIPULATION, ALL PROGRAMS IN THIS SYSTEM
C       USE A PARTICULAR CODE, IN WHICH ALL CHARACTERS ARE REPRESENTED
C       BY INTEGER NUMBERS. THE TABLE BELOW IS THE MAPPING BETWEEN
C       ACTUAL (INPUT/OUTPUT) CHARACTERS AND THE INTERNAL CODE.
C
C     ICC     CHAR    HEX    DEC         ICC    CHAR     HEX    DEC
C      1       BLANK   40     64           25     N        D5     213
C      2       0       F0     240          26     O        D6     214
C      3       1       F1     241          27     P        D7     215
C      4       2       F2     242          28     Q        D8     216
C      5       3       F3     243          29     R        D9     217
C      6       4       F4     244          30     S        E2     226
C      7       5       F5     245          31     T        E3     227
C      8       6       F6     246          32     U        E4     228
C      9       7       F7     247          33     V        E5     229
C     10       8       F8     248          34     W        E6     230
C     11       9       F9     249          35     X        E7     231
C     12       A       C1     193          36     Y        E8     232
C     13       B       C2     194          37     Z        E9     233
C     14       C       C3     195          38     #        5B     91
C     15       D       C4     196          39     =        7E     126
C     16       E       C5     197          40     .        4B     75
C     17       F       C6     198          41     <        61     97
C     18       G       C7     199          42     (        4D     77
C     19       H       C8     200          43     )        5D     93
C     20       I       C9     201          44     >        4E     78
C     21       J       D1     209          45     -        6C     96
C     22       K       D2     210          46     :        7D     125
C     23       L       D3     211          47     *        5C     92
C     24       M       D4     212          48     ,        6B     107
C
C        THE MAIN PROGRAM IS SIMPLY A DRIVER WHICH CALLS THE  DATA
C       GENERATOR, THE TABLE GENERATOR OR THE ASSEMBLER ACCORDING
C       TO THE VALUE OF THE NUMBER READ ON THE FIRST CARD.
C
C       INITIALIZE VARIABLES.
C
        CALL INIT
        READ(5,100) I
  100   FORMAT(I1)
        GO TO (1, 2, 3),I
    1   CALL DATGEN
        GO TO 999
    2   CALL GENER
        GO TO   999
    3   CALL ASMAIN
  999   STOP
        END
```

```
      BLOCK DATA
      INTEGER OTRAN, OBUFF, CEP, OUTREC, OUTLIM , FWIDTH
      COMMON /INFO/ NF, FWIDTH (16), ISTART(16)
      COMMON /MEMO/ MEMORY(2000), MEMBCT, MEMTOP, NDIV
      COMMON /IOUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /TRANS/ ITRAN(256), OTRAN(64)
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /MACHIN/ NBITS
      COMMON /RECSZE/ OUTREC, OUTLIM
      DATA OUTREC /80 /
      DATA OBP/1/ , IBP /81/
      DATA  NBITS /24/
      DATA NREAD/5/, NPRINT/6/, NPUNCH/6/
      DATA ITRAN/256 * 1H /
      DATA OTRAN/1H ,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,1HA,1HB,1HC
     1,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1
     1HT,1HU,1HV,1HW,1HX,1HY,1HZ,1H$,1H=,1H.,1H<,1H(,1H),1H>,1H-,1H:,1H*
     2,1H,,1H/,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
     31H /
C
C     THE OUTPUT OF THE TABLE GENERATOR SHOULD BE INSERTED RIGHT AFTER
C     THIS COMMENT.
C
      DATA MEMORY / 2000 * 0 /
      END
```

```
      SUBROUTINE DATGEN
C
C                   DATA GENERATOR MAIN
C
C
C     THIS SUBROUTINE HELPS PREPARING THE INPUT CARDS TO THE TABLE
C     GENERATOR.
C
C
      INTEGER FWIDTH, CONV, ACCUM, ACCLEN, TYPE
      COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
      COMMON /ICUNIT/ NREAD, NPRINT, NPUNCH
      DIMENSION  FWIDTH(16)
C
C     FROM NOW ON OUTPUT IS ON THE CARDPUNCH.
C
      NPRINT = 7
      NPUNCH = 7
      CALL PAD(1, 4, 1)
      CALL WRITEL(0, NPUNCH)
C
C     GET THE NUMBER OF FIELDS IN EACH WORD .
C
      CALL SCAN
      NF = CONV(M)
C
C     GET THE NUMBER OF BITS IN EACH FIELD.
C
          DO  400 I = 1, NF
          CALL SCAN
          FWIDTH(I) = CONV(M)
  400     CONTINUE
C
C     OUTPUT THE NUMBER OF FIELDS IN EACH WORD.
C
      CALL CONOUT(1, -2, NF, 10)
      CALL PAD(1, 48, 1)
C
C     OUTPUT THE NUMBER OF   BITS IN EACH FIELD.
C
          DO  402 I = 1, NF
          CALL CONOUT(1, -2, FWIDTH(I), 10)
          CALL PAD(1,48,1)
  402     CONTINUE
      CALL WRITEL(0,NPUNCH)
C
C     GENERATE AND OUTPUT THE CODE CORRESPONDING TO THE
C     'NEXT BASIC ADDRESS' FIELD.
```

```
C
      NFW = FWIDTH(1)
      LIM = 2 **(NFW + 1)
      I = 0
      GO TO 3
    4 CALL PAD(1, 48, 1)
    3 J = I / 2
      CALL CONOUT(1, -3, I, 10)
      CALL PAD(1, 48, 1)
      CALL CONOUT(1,NFW,J,2)
      I = I + 1
      IF ( I .LT. LIM) GO TO 4
      CALL PAD(1, 40, 1)
      CALL WRITEL(0,NPRINT)
C
C     READ EACH MNEMONIC, GENERATE ITS CODE AND CUTPUT BOTH.
C
          DO  401  K = 2, NF
          I = 0
          GO TO 1
    5     CALL PAD(1, 48, 1)
    1     CALL SCAN
          J = ACCLEN - 1
          CALL FORM(1, ACCUM, 1, J, 32)
          CALL PAD(1, 48, 1)
          CALL CONOUT (1, FWIDTH(K), I, 2)
          I = I + 1
          IF (TYPE .NE. 3) GC TO 5
    2     CALL PAD(1,40,1)
          CALL WRITEL(0,NPUNCH)
  401     CONTINUE
      STOP
      END
```

```
      SUBROUTINE GENER
C
C                  TABLE GENERATOR MAIN
C
      INTEGER OUTMSG, FWIDTH, TYPE, ACCLEN, ACCUM, OBUFF, OBP
      INTEGER GET, CONV
      INTEGER OUTREC, OUTLIM
      DIMENSION OUTMSG(34)
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /ICUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /MEMO/ MEMORY(2000), MEMBCT, MEMTCP, NDIV
       COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
      COMMON /RECSZE/ OUTREC, OUTLIM
      COMMON /INFO/ NF, FWIDTH (16), ISTART(16)
      DATA OUTMSG/15, 12, 31, 12, 1, 24, 16, 24,          26, 29, 36, 1,
     1  49, 20, 30, 31, 12, 29, 31, 49 , 25, 17, 1, 49, 17, 34, 20,
     215, 31, 19, 1, 49 /
C
C     READ NUMBER OF FIELDS AND NUMBER CF BITS CONTAINED IN EACH FIELD.
C
      CALL SCAN
      NF = CONV(M)
           DO 0400 I = 1, NF
           CALL SCAN
           FWIDTH(I) = CONV(M)
 0400      CONTINUE
      NPRINT = 7
      NPUNCH = 7
      CALL PAD(1, 1, 6)
      CALL FORM (1, OUTMSG, 1, 5, 34)
      CALL FORM(1, OUTMSG, 21, 24, 34)
      CALL CONOUT(1, -3, NF, 10)
      CALL PAD(1, 49, 1)
      CALL PAD(1, 48, 1)
      CALL FORM(1, OUTMSG, 25, 32, 34)
           DO   406 I = 1, NF
           CALL CONOUT(1, -3, FWIDTH(I), 10)
           IF (I .EQ. NF) GO TO 2
           CALL PAD(1, 48, 1)
  406      CONTINUE
    2 CALL PAD(1, 49, 1)
      CALL WRITEL(0, NPUNCH)
C
C     READ ALL LEGAL INPUTS AND CORRESPONDING CCDE TO BE PRODUCED,
C     FIELD BY FIELD.
C
           DO 0401 I = 1, NF
C
```

```
C      SAVE THE LOCATION WHERE THE TABLE FOR THIS FIELD STARTS.
C
            ISTART(I) = MEMBOT
C
C      MAKE THE EMPTY INPUT CORRESPOND TO CODE 0.
C
            CALL PUT(MEMBOT,1)
            K = FWIDTH(I)
                  DO 0405 J = 1, K
                  CALL PUT(MEMBOT, 2)
 0405             CONTINUE
C
C      GET THE NEXT ELEMENT.
C
 0001       CALL SCAN
C
C      IF THE INPUT CONTAINS A '.' IN THE FIRST PLACE, THAT IS ALL
C      THERE IS FOR THE CURRENT FIELD.
C
       IF (TYPE .EQ. 3) GO TO 0401
C
C      STORE THE LENGTH OF THIS ELEMENT.
C
            CALL PUT(MEMBOT, ACCLEN)
C
C      STORE THE ELEMENT IN 'MEMORY',
C
       K = ACCLEN - 1
            DO 0402 J = 1, K
                  CALL PUT(MEMBOT, ACCUM(J))
 0402             CONTINUE
C
C      GET THE CORRESPONDING CODE.
C
            CALL SCAN
       IF (TYPE .EQ. 3) GO TO 0401
C
C      STORE IT INTO 'MEMORY'.
C
            K = ACCLEN - 1
            DO 0403 J = 1, K
                  CALL PUT(MEMBOT, ACCUM(J))
 0403             CONTINUE
            GO TO 1
 0401       CONTINUE
C
C      NOW DUMP 'ISTART'.
C
```

```
      OUTREC = 72
      OUTLIM = 73
      CALL PAD(1, 1, 6)
      CALL FORM(1, OUTMSG, 1, 5, 34)
      CALL FORM(1, OUTMSG, 14, 20, 34)
      I = 1
 0006 CALL CONOUT(1, -4, ISTART(I), 10)
      I = I + 1
      I F(I .GT. NF) GO TO 7
      CALL PAD(1, 48, 1)
      IF (OBP .LT. 68) GO TO 6
      CALL WRITEL(0, NPUNCH)
      CALL PAD(1, 1, 5)
      CALL PAD(1, 3, 1)
      GO TO 6
 0007 CALL PAD(1, 48, 1)
      CALL CONOUT(1, 5,  4000, 10)
      CALL PAD(1, 49, 1)
      CALL WRITEL(0, NPUNCH)
C
C     DUMP 'MEMORY'.
C
      CALL PAD(1, 1, 6)
      CALL FORM(1, OUTMSG, 1, 5, 34)
      CALL FORM(1, OUTMSG, 6, 13, 34)
      LIM = MEMBOT / NDIV + 1
      I = 1
 0004 CALL CONOUT(1,-10, MEMORY(I), 10)
      I = I  + 1
      I F( I .GT. LIM) GO TO 5
      CALL PAD (1, 48, 1)
      IF (OBP .LT. 62) GO TO 4
      CALL WRITEL(0, NPUNCH)
      CALL PAD(1, 1, 5)
      CALL PAD(1, 3, 1)
      GOTO 4
 0005 CALL PAD(1, 49, 1)
      CALL WRITEL(0, NPUNCH)
      RETURN
      END
```

```
      SUBROUTINE ASMAIN
C
C                ASSEMBLER MAIN
C
      INTEGER DIGIT
      INTEGER TYPE, VALUE, CONV, FIELD, TEMP1, TEMP2, FWIDTH, GET,
     1  ACCLEN, ACCUM, CODE, ZERO
      DIMENSION ISTART(16), FWIDTH(16), CODE(16), ZERO(1)
      COMMON /MEMO/ MEMORY(2000), MEMBCT, MEMTOP, NDIV
       COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
      COMMON /ICUNIT/ NREAD, NPRINT, NPUNCH
      DATA ZERO /1/
       DATA ISTART/   1, 504, 618,   843 /
      DATA FWIDTH / 5, 4, 7 /
C
C     INITIALIZE
C
      MEMTOP = 4000
      KP = 7
      KODE = 0
      L = 0
      LABEL = - 1
      WRITE(6, 363)
 363  FORMAT(1H1)
C
C     GET THE NEXT ELEMENT.
C
 0004 CALL SCAN
C
C     SEE 'SCAN' FOR DEFINITION OF 'TYPE' VALUES.
C
C
      GO TO (1, 2, 2, 12), TYPE
C
C     THE ELEMENT IS A <LABEL>. CHECK FOR SEQUENCE GAP.
 0001 VALUE = CONV(M)
 0013 LABEL = LABEL + 1
      IF (VALUE .EQ. LABEL) GO TO 0005
C
C     THERE IS A SEQUENCE GAP.
C
      CALL PUT(-MEMTOP,0)
      CALL PUT(-MEMTOP,0)
      GO TO 0013
C
C     NOW GET THE CONTENTS OF FIELD 1.
C
C
```

```
 0005 FIELD = 1
      GO TO 0004
C
C     THE ELEMENT IS AN IDENTIFIER  .FIND IT IN THE LIST.
C
 0002 I = ISTART(FIELD)
      TEMP1 = FWIDTH(FIELD)
 0007 TEMP2 = GET(I)
      IF (ACCLEN .EQ. TEMP2) GO TO 0006
 0011 I = I + TEMP2 + TEMP1
      IF (I .LT. ISTART(FIELD+1)) GO TO 007
      CALL ERROR(39)
      GC TO 0012
C
C     THE LENGTHS AGREE. CHECK ALL CHARACTERS, CNE AT A TIME.
C
 0006 J = 1
      K = I + 1
      IF (ACCLEN .EQ. 1) GO TO 10
 0009 IF (ACCUM(J) .NE. GET(K)) GO TO 0011
      J = J + 1
      K = K + 1
      IF (J .LT. ACCLEN) GO TC 0009
C
C
C
C     WE FOUND IT. GET THE CORRESPONDING CODE.
C
 0010 L = 1
      LIM = K + TEMP1 - 1
            DO 403 KI = K,LIM
            KODE = KODE + (GET(KI) - 2) * 2 ** KP
            KP = KP - 1
            I F(KP .GE. 0) GO TO 403
            CALL PUT(-MEMTOP,KODE)
            IF (MEMTOP .LE. MEMBOT) CALL ERROR (27)
            KP = 7
            KODE = 0
 0403       CONTINUE
      IF (TYPE .LE. 2) FIELD = FIELD + 1
      GO TO 4
   12 CALL PUNCH
 0999 RETURN
      END
```

```
      SUBROUTINE INIT
C
C     THIS SUBPROGRAM INITIALIZES ALL THE TABLES AND VARIABLES FOR
C     USE OF THE SYSTEM.
C
C     MEANING OF VARIABLES:
C     MEMORY   VIRTUAL MEMORY USED BY THE SYSTEM. EACH COMPUTER WORD
C              IN 'MEMORY' IS SUBDIVIDED IN 'NDIV' SUBDIVISIONS.
C     OUTREC   SIZE OF THE OUTPUT RECORD
C     OUTLIM   EQUAL TO OUTREC + 1 IS USED AS TESTING PARAMETER.
C     OTRAN    VECTOR OF LENGTH 64 CONTAINS ALL THE CHARACTERS THAT
C              WILL BE RECOGNIZED BY THE SYSTEM.
C     NDIV     NUMBER OF DIVISIONS WITHIN A COMPUTER WORD FOR USE
C              AS VIRTUAL STORAGE.
C     NBITS    NUMBER OF BITS IN ONE WORD (THIS VALUE IS TO BE MODIFIED
C              FOR USE ON DIFFERENT COMPUTERS IF MAXIMUM EFFICIENCY
C              OF MEMORY USE IS TO BE OBTAINED.
C     MEMBOT   POINTER TO THE HIGHER LOCATION IN VIRTUAL MEMORY NOT YET
C              USED.
C     ITRAN    VECTOR OF LENGTH 256 CONTAINS ALL THE CODES FOR THE
C              CHARACTERS RECOGNIZED BY THE SYSTEM.
C     ICON     SUBPROGRAM THAT, GIVEN A CHARACTER , RETURNS ITS CODE.
C
      INTEGER OUTREC, OUTLIM, OTRAN
      COMMON /MEMO/ MEMORY(2000), MEMBOT, MEMTOP, NDIV
      COMMON /TRANS/ ITRAN(256), OTRAN(64)
      COMMON /MACHIN/ NBITS
      COMMON /RECSZE/ OUTREC, OUTLIM
C
      OUTLIM = OUTREC + 1
      NDIV = (NBITS - 1) / 8
      MEMBOT = 1
C
C     SET UP ITRAN (INPUT TRANSLATOR TABLE).
C
          DO 0400 I = 1, 49
          J = OTRAN (I)
          J = ICON (J)
          ITRAN(J) = I
0400      CONTINUE
      RETURN
      END
```

```
      INTEGER FUNCTION ICON(N)
C
C     THIS SUBPRCGRAM CCDES CHARACTERS INTO THE INTERMEDIATE CODE (AN
C     INTEGER BETWEEN 1 AND 256).
C
C     MEANING OF VARIABLES:
C
C     ICON        INPUT CCNVERSION.
C     N           CHARACTER (EBCDIC, ASCII, ETC.) TC BE CODED.
C     OTRAN       VECTOR OF LENGTH 64 USED TO MAP ALL LEGAL INPUT CHA-
C                 RACTERS INTO A SET OF INTEGER NUMBERS LESS THAN OR
C                 EQUAL TO 64.
C
      INTEGER OTRAN
      CCMMON /TRANS/ ITRAN(256), OTRAN(64)
C
C     FIND 'N' IN 'CTRAN'.
C
         DO 2400 I = 1, 64
         IF (N .EQ. OTRAN(I)) GC TO 2001
 2400    CONTINUE
C
C     'N' WAS NOT FOUND IN 'OTRAN' THEREFORE MAKE IT EQUAL TO BLANK.
C
      I = 1
2001  ICON = I
      RETURN
      END
```

```
      INTEGER FUNCTION CONV(PREC)
C
C     THIS SUBPROGRAM TAKES A NUMBER IN THE ACCUMULATOR ('ACCUM') AND
C     CONVERTS IT TO THE NORMAL BINARY MACHINE REPRESENTATION.
C
      INTEGER  PREC, DIGIT
      INTEGER ACCUM, ACCLEN, TYPE
       COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
C
C     INITIALIZE
C
      CONV = 0
      PREC = 0
      I = ACCLEN - 1
C
C     GET THE LEAST SIGNIFICANT DIGIT AND DECODE IT.
C
    1 DIGIT = ACCUM(I) - 2
      IF (DIGIT .GT. 9) CALL ERROR(14)
C
C     UPDATE 'CONV'
C
      CONV = CONV + DIGIT * 10 ** (ACCLEN - 1 - I)
      PREC = PREC + 1
      I = I - 1
C
C     IF THERE ARE MORE DIGITS, DO IT AGAIN
C
      IF (I .NE. 0) GO TO 1
      RETURN
      END
```

```
      SUBROUTINE CONOUT(CC,FIELDW,VAL,BASE)
C
C     THIS SUBPROGRAM PLACES THE VALUE OF 'VAL' INTO THE OUTPUT BUFFER
C     IN A FIELD WIDTH OF 'FIELDW' USING THE RADIX 'BASE'. IF 'FIELDW'
C     IS NEGATIVE, SUPPRESS LEADING ZEROS.
C
C
      INTEGER OBUFF, OBP, OUTREC, OUTLIM
      INTEGER TEMP, CC, FIELDW, VAL, BASE, PTR
      COMMON /IOUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /RECSZE/ OUTREC, OUTLIM
C
      TEMP = IABS(VAL)
C
C     IF 'CC' = 0 DUMP CURRENT 'OBUFF' AND START A NEW LINE.
C
      IF (CC .EQ. 0) CALL WRITEL(0,NPRINT)
C
C     SET A POINTER TO THE END OF THE FIELD.
C
      PTR = OBP + IABS(FIELDW) - 1
C     IF THE NUMBER WILL NOT FIT IN THIS RECORD, START A NEW ONE.
C
      IF (PTR .LT. OUTLIM) GO TO 4
      CALL WRITEL(0, NPRINT)
      PTR = IABS(FIELDW)
C
C     COPY THE LOWER LIMIT OF THE FIELDW INTO 'LOWER'.
C
    4 LOWER = OBP
C
C     SET 'OBP' TO THE FIRST SPACE AVAILABLE AFTER THE NUMBER
C
      OBP = PTR + 1
C
C     GET THE RIGHTMOST DIGIT FROM 'VAL'.
C
    1 NDIG = MOD (TEMP, BASE)
C
C     CODE IT INTO INTERMEDIATE CODE.
C
      NDIG = NDIG + 2
C
C     CHECK TO SEE IF THERE IS STILL ROOM IN THE FIELD.
C
      IF (PTR .LT. LOWER) GO TO 2
C
```

```
C       STORE IT INTO 'OBUFF'.
C
        OBUFF(PTR) = NDIG
        PTR = PTR - 1
C
C       DROP THE RIGHTMOST DIGIT OF 'VAL'.
C
        TEMP = TEMP / BASE
C
C       WAS THIS THE MOST SIGNIFICANT DIGIT OF 'VAL' ?
C
        IF (TEMP .NE. 0) GO TO 1
        IF (PTR .LT. LOWER) GO TO 3
C
C       FILL THE LEADING SPACES WITH ZEROS (IF 'FIELDW' < 0)
C       OR BLANKS.
C
        KAR = 2
        IF (FIELDW .LT. 0) KAR = 1
        L = PTR
              DO  400 PTR = LOWER, L
              OBUFF(PTR) = KAR
  400         CONTINUE
        GC TO 3
    2   CALL ERROR(20)
    3   RETURN
        END
```

```
      SUBROUTINE PAD(CC, CHR, NCHRS)
C     THIS SUBPROGRAM PLACES THE CHARACTER 'CHR' REPEATED 'NCHRS' TIMES
C     INTO THE OUTPUT BUFFER.
C     MEANING OF VARIABLES:
C
C     CC        CARRIAGE CONTROL. IF = 0 DUMP CURRENT BUFFER FIRST.
C                                 IF = 1 APPEND TO CURRENT BUFFER.
C     CHR       CHARACTER TO BE INSERTED INTO 'OBUFF'.
C     OBUFF
C     OBP
C     NCHRS     NUMBER OF TIMES 'CHR' IS TO BE REPEATED.
      INTEGER OBUFF, OBP, OUTREC, OUTLIM
      INTEGER CC, CHR
      COMMON /IOUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /RECSZE/ OUTREC, OUTLIM
C
      IF (CC .EQ. 0) CALL WRITEL(0,NPRINT)
C
          DO 7400 I = 1, NCHRS
      IF (OBP .EQ. OUTLIM) CALL WRITEL(0,NPRINT )
          OBUFF(OBP) = CHR
          OBP = OBP + 1
 7400     CONTINUE
      RETURN
      END
```

```
      SUBROUTINE FORM(CC, CHARS, START, FINISH, LEN)
C     THIS SUBPROGRAM PLACES THE CHARACTERS FROM 'CHARS' INTO THE OUTPUT
C     BUFFER STARTING AT 'CHARS(START)' AND ENDING AT 'CHARS(FINISH)'.
C
C     MEANING OF VARIABLES:
C
C     CC         CARRIAGE CONTROL.IF 'CC' = 0 DUMP CURRENT 'OBUFF' AND
C                START A NEW LINE.
C                                 IF CC=1 APPEND TO CURRENT 'OBUFF'.
C     CHARS      VECTOR OF LENGTH 'LEN' WHICH CONTAINS THE CHARACTERS TO
C                BE PLACED IN 'OBUFF'.
C     START      POINTER TO THE FIRST MEMBER OF 'CHARS' TO GET OUT.
C     FINISH     POINTER TO THE LAST MEMBER OF 'CHARS' TO GET OUT.
C     LEN        LENGTH. DIMENSION OF 'CHARS'.
C
      INTEGER OBUFF, OBP, OUTREC, OUTLIM
      INTEGER CC, CHARS, START, FINISH
      COMMON /ICUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /RECSZE/ OUTREC, OUTLIM
      DIMENSION CHARS(LEN)
C
C     IF 'CC' = 0 START A NEW LINE.
C
      IF (CC .EQ. 0) CALL WRITEL(0, NPRINT)
C
C     PLACE DESIRED INFORMATION INTO 'OBUFF'.
C
      I = START
    1 OBUFF(OBP) = CHARS(I)
      OBP = OBP + 1
      IF (OBP .EQ. OUTLIM) CALL WRITEL(0,NPRINT)
      I = I + 1
      IF (I .LE. FINISH) GO TO 1
      RETURN
      END
```

```
      SUBROUTINE WRITEL(NLINE, NUNIT)
C
C     THIS SUBPROGRAM OUTPUTS THE CURRENT OUTPUT BUFFER VIA I/O UNIT
C     'NUNIT' AND APPENDS 'NLINE' BLANK RECORDS.
C
C     MEANING OF VARIABLES:
C
C     NLINE      NUMBER OF LINES (OR RECORDS) TO BE INSERTED AFTER
C                OUTPUTTING THE CURRENT OUTPUT BUFFER.
C     NUNIT      LOGICAL NUMBER OF OUTPUT UNIT TO BE USED.
C
      INTEGER OTRAN, OBUFF, OBP, OUTREC,OUTLIM
      INTEGER TEMP
      COMMON /IOUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /TRANS/ ITRAN(256), OTRAN(64)
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /RECSZE/ OUTREC, OUTLIM
C
C     IF 'OBUFF' IS NOT COMPLETELY FULL, PAD WITH BLANKS.
C
      IF (OBP .EQ. OUTLIM) GO TO 6001
      TEMP = OUTLIM - OBP
      CALL PAD (1, 1, TEMP)
C
C     DECODE 'OBUFF'.
C
 6001     DO 6400 I = 1, OUTREC
          OBUFF(I) = OTRAN(OBUFF(I))
 6400     CONTINUE
C
C     NOW DUMP IT.
C
      IF (NUNIT .NE. 7) GO TO 6004
      WRITE(NUNIT,6103) (OBUFF(I),I = 1, OUTREC)
 6103 FORMAT (120A1)
      GO TO 6003
 6004 WRITE(NUNIT,6100)(OBUFF(I),I=1,OUTREC)
 6100 FORMAT(1X, 120A1)
C
C     RESET OBP.
C
 6003 OBP = 1
C
C     SEE IF THERE  ARE BLANK RECORDS TO BE INSERTED.
C
      IF (NLINE .EQ. 0) GO TO 6002
          DO 6401 I = 1, NLINE
          WRITE (NUNIT, 6101)
```

```
6101       FORMAT(1H )
6401       CONTINUE
6002  RETURN
      END
```

```
      SUBROUTINE SCAN
C
C     THIS SUBROUTINE :
C     A) GETS THE NEXT ELEMENT IN THE INPUT STREAM (CHARACTER BY
C        CHARACTER, IGNORING BLANKS).
C     B) STORES IT, ALREADY CODED INTO INTERMEDIATE CODE (AN INTEGER
C        BETWEEN 1 AND 64  FOR EACH CHARACTER) INTO AN ACCUMULATOR.
C     C) GETS THE NEXT SYMBOL, WHICH SHOULD BE ONE OF THE SPECIAL CHA-
C        RACTERS USED AS <DELIMITER>
C     D) RETURNS THE ACCUMULATOR AND THE TYPE OF  ELEMENT IN IT TO THE
C        CALLING PROGRAM.
C     E) ISSUES ERROR  MESSAGES WHENEVER APPRORIATE.
C
C     MEANING OF VARIABLES:
C
C     NC          NEXT CHARACTER  IN THE INPUT STREAM
C     ACCUM       ACCUMULATOR. VECTOR OF LENGTH 32 USED TO HOLD THE
C                 ELEMENT BEING ANALYZED.
C     ACCLEN      ACCUMULATOR LENGTH. POINTER TO 'ACCUM' TELLS HOW
C                 LONG IS THE ELEMENT SO FAR.
C     TYPE        IF TYPE = 1 THEN THE ELEMENT IS A LABEL.
C                 IF TYPE = 2 THEN THE ELEMENT IS AN IDENTIFIER.
C                 IF TYPE = 3 THEN THE ELEMENT IS THE LAST IDENTIFIER
C                 IN THIS STATEMENT.
C                 IF TYPE = 4 THIS IS THE LAST CARD IN THE PROGRAM.
C
      INTEGER ACCUM, ACCLEN, TYPE
      INTEGER GNC
       COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
C
C     RESET 'ACCLEN'.
C
      ACCLEN = 1
C
C     GET A CHARACTER FROM INPUT STREAM.
C
3002  NC = GNC(0)
C
C     IS IT AN ALPHANUMERIC CHARACTER ('$' INCLUSIVE) ? IF NOT,
C     GO TO 3003.
C
      IF (NC .GT. 38) GO TO 3003
C
C     IF 'NC' = 0 IT IS TIME TO STOP.
C
      IF (NC .NE. 0) GO TO 3001
      TYPE = 4
      GO TO 3009
```

```
C
C     WE FOUND AN IDENTIFIER. COPY IT INTO 'ACCUM'.
C
3001  ACCUM(ACCLEN) = NC
      ACCLEN = ACCLEN + 1
      IF (ACCLEN .LT. 33) GO TO 3002
      CALL ERROR(18)
      GC TO 3009
C
C     WE FOUND A SPECIAL CHARACTER. THERE ARE FOUR LEGAL CASES:
C     A) A COMMA(48) MEANING THAT WE HAVE AN IDENTIFIER IN 'ACCUM'.
C     B) A COLON (46) MEANING THAT WE HAVE A LABEL IN 'ACCUM'.
C     C) A PERIOD (40) MEANING THAT THE IDENTIFIER IN 'ACCUM' IS THE
C        LAST IN THIS STATEMENT.
C     D) A 'LESS THAN' SIGN (41) MEANING THAT WHAT FOLLOWS IS A COMMENT.
C
3003  IF (NC .NE. 48) GC TO 3004
C
C     WE FOUND A COMMA ; SET 'TYPE' = 2 AND RETURN.
C
      TYPE = 2
      GC TO 3009
3004  IF (NC .NE. 46) GO TO 3005
C
C     WE FOUND A COLON ; SET 'TYPE' = 1 AND RETURN.
C
      TYPE = 1
      GC TO 3C09
3005  IF ( NC .NE. 40) GO TO 3006
      TYPE = 3
      GO TO 3009
3006  IF (NC .NE. 41) GO TO 3007
3011  NC = GNC(0)
      IF (NC .NE. 44) GO TO 3011
      GC TO 3002
3007  CALL ERROR (19)
3009  RETURN
      END
```

```
      INTEGER FUNCTION GNC(Q)
C
C     THIS SUBPROGRAM GETS THE NEXT NON-BLANK CHARACTER IN THE INPUT
C     STREAM AND CODES IT INTO THE INTERMEDIATE CODE (AN INTEGER BETWEEN
C     1 AND 256).
C     ALSO 'ECHO CHECKS' ALL CARDS IMMEDIATELY AFTER READING.
C     WHEN A STAR ('*') IS  ENCOUNTERED IN THE FIRST COLUMN OF A CARD IT
C     IS INTERPRETED AS END OF FILE AND THE VALUE RETURNED BY 'GNC' IS
C     ZERO.
C
C     MEANING OF VARIABLES:
C
C     GNC         GET NEXT CHARACTER.
C     Q           DUMMY ARGUMENT. NOT USED BUT NEEDED BECAUSE 'GNC' IS
C                 A FUNCTION.
C     IBUFF       INPUT BUFFER. VECTOR OF LENGTH 80 USED TO HOLD ONE
C                 CARD IMAGE.
C     IBP         INPUT BUFFER POINTER. NEXT CARD COLUMN TO BE LOOKED AT.
C     ITRAN       INPUT TRANSLATOR. VECTOR OF LENGTH 256 USED AS A TABLE
C                 OF CORRESPONDENCE BETWEEN INPUT STREAM SYMBOLS AND THEIR
C                 REPRESENTATION IN INTERMEDIATE CODE (AN INTEGER BETWEEN
C                 1 AND 256).
C     NC          NEXT CHARACTER. USED TO HOLD TEMPORARY VALUES OF 'GNC'.
C
      INTEGER OTRAN, OBUFF, OBP
      COMMON /IOUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /TRANS/ ITRAN(256), OTRAN(64)
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
C
C     IF WE HAVE USED THIS ENTIRE CARD, GET A NEW ONE.
C
1003      IF (IBP .LE. 80 ) GO TO 1001
C
C     READ IN A NEW CARD, RESET 'IBP' AND ECHO-CHECK THE CARD.
C
      READ(NREAD, 1100) IBUFF
1100  FORMAT(80A1)
      WRITE(6, 1101) IBUFF
 1101 FORMAT(1X, 80A1)
      IBP = 1
C
C     GET A CHARACTER FROM 'IBUFF'.
C
1001  NC = IBUFF(IBP)
      IBP = IBP + 1
C
C     CODE IT INTO INTERMEDIATE CODE.
C
```

```
      NC = ICON(NC)
      NC = ITRAN(NC)
C
C     GET RID OF IT IF IT IS A BLANK.
C
      IF (NC .EQ. 1) GO TO 1003
C
C     THE FIRST CHARACTER ON A CARD GETS SPECIAL ATTENTION, AS IT COULD
C     BE AN END OF FILE MARK.
C
      IF (IBP .NE. 2 ) GO TO 1002
C
C     IF THE FIRST CHARACTER IN THE CARD IS A '*' 'GNC' MUST BE SET
C     TO ZERO.
C
      IF (NC .EQ. 47) NC = 0
1002  GNC = NC
      RETURN
      END
```

```
      INTEGER FUNCTION GET(LOC)
C     THIS SUBPROGRAM GETS THE POSITIVE INTEGER NUMBER , LESS THAN
C     256 , STORED IN VIRTUAL MEMORY ADDRESS 'LOC'.
C
C     MEANING OF VARIABLES:
C
C     ADD        REAL MEMORY ADDRESS WHICH CONTAINS 'LOC'.
C     TEMP       AUXILIARY VARIABLE USE TO HOLD TEMPORARY VALUES.
C     TEMP1      AUXILIARY VARIABLE USE TO HOLD TEMPORARY VALUES.
C
      INTEGER ADD, TEMP, TEMP1
      COMMON /MEMO/ MEMORY(2000), MEMBCT, MEMTOP, NDIV
C
C     GET THE REAL MEMORY ADDRESS CORRESPONDING TO 'LOC'.
C
      ADD = (LOC - 1) / NDIV + 1
C
C     COPY ITS CONTENTS TO MAKE EXECUTION FASTER.
C
      TEMP = MEMORY(ADD)
C
C     LOCATE 'LOC' INSIDE THE ADDRESS 'ADD'.
C
      TEMP1 = LOC - ((ADD - 1) * NDIV)
      TEMP1 = NDIV - TEMP1
      TEMP1 = 256 ** TEMP1
      TEMP = TEMP / TEMP1
      GET = MOD(TEMP, 256)
      RETURN
      END
```

```
      SUBROUTINE PUT(LOC, VAL)
C
C     THIS SUBPROGRAM STORES AN INTEGER NUMBER 'VAL' (BETWEEN 0 AND 255)
C     INTO 8 BITS OF A COMPUTER WORD
C     THIS IS DONE TO SAVE STORAGE SPACE.
C
C     MEANING OF VARIABLES:
C
C     LOC         LOCATION IN THE 'VIRTUAL MEMORY' WHERE THE VALUE IS TO
C                 BE STORED. IF POSITIVE, INCREMENT 'MEMBOT' . IF NEGATIVE
C                 DECREMENT MEMTOP '.
C     VAL         INTEGER VALUE TO BE STORED IN 'LOC'.
C     MEMBOT      MEMORY BOTTOM. POINTER TO THE LOWEST NUMBERED VIRTUAL
C                 ADDRESS AVAILABLE.
C     MEMTOP      MEMORY TOP. POINTER TO THE HIGHEST NUMBERED VIRTUAL
C                 ADDRESS AVAILABLE.
C     INCR        INCREMENT. ADDS 1 TO 'MEMBOT' OR SUBTRACTS 1 FROM
C                 'MEMTOP' DEPENDING UPON THE SIGN OF 'LOC'.
C     ADD         REAL MEMORY ADDRESS WHICH CONTAINS 'LOC'.
C     TEMP        AUXILIARY VARIABLE TO HOLD TEMPORARY VALUES.
C     TEMP1       AUXILIARY VARIABLE TO HOLD TEMPORARY VALUES.
C     TEMP2       AUXILIARY VARIABLE TO HOLD TEMPORARY VALUES.
C     NDIV        NUMBER OF DIVISIONS  (VIRTUAL MEMORY ADDRESSES)
C                 INSIDE A REAL MEMORY WORD.
C     SAV         CONTENTS OF REAL MEMORY WORD TO THE RIGHT OF 'LOC'.
C
      INTEGER VAL, ADD, TEMP, TEMP1, TEMP2, SAV
      COMMON /MEMO/ MEMORY(2000), MEMBOT, MEMTOP, NDIV
C
C     SET 'INCR' TO THE PROPER VALUE.
C
      INCR = 1
      IF (LOC .GT. 0) GO TO 5001
      LOC = -LOC
      INCR = - INCR
C
C     COMPUTE THE ACTUAL MEMORY WORD ADDRESS.
C
 5001 ADD = (LOC - 1) / NDIV + 1
      I F(ADD .GT.NDIV * 2000) CALL ERROR(37)
C
C     COPY THE CURRENT CONTENTS OF 'MEMORY(ADD)' TO SPEED UP COMPUTATION
C
      TEMP = MEMORY (ADD)
C
C     FIND THE POSITION OF 'LOC' INSIDE THE WORD. IT WILL BE AN INTEGER
C     BETWEEN 1 AND NDIV.
C
```

```
      TEMP1 = NDIV - (LOC - NDIV * (ADD - 1))
C
C     SAVE THE VALUES TO THE RIGHT OF 'LCC'.
C
      TEMP1 = 256 ** TEMP1
      SAV = MOD (TEMP, TEMP1)
      TEMP2 = TEMP1 * 256
      TEMP = (TEMP / TEMP2 * TEMP2 + VAL ) * TEMP1 + SAV
      MEMORY (ADD)=TEMP
      LCC = LCC + INCR
      RETURN
      END
```

```
      SUBROUTINE PUNCH
C
C     THIS SUBROUTINE, CALLED AT THE END OF THE ASSEMBLER, DUMPS
C     THE CONTENTS OF 'MEMORY' INTO A PAPER TAPE IN A FORMAT SUITABLE
C     TO PROGRAM A ROM. IT SHOULD BE REVISED WHENEVER CHANGES ARE MADE
C     THE TAPE PUNCH OR TO THE ROM PROGRAMMING SYSTEM.
C
      INTEGER RUBOUT, B, P, F, GET
      DATA RUBOUT/177/, B/102/, P/120/, N/116/, F/106/
      KA = 4000
   15      DO  400 I = 1,25
  400      WRITE(7,100) RUBOUT
  100 FORMAT(I1)
           DO  404 I = 1,256
           WRITE(7,100)B
            KODE = GET(KA)
                DO 405 J = 1, 8
                DIGIT = MOD(KODE, 2)
                KODE = KODE / 2
                IF (DIGIT) 1,2,3
    1 CALL ERROR(27)
                RETURN
    2 WRITE  (7,100) N
      GO TO  405
    3 WRITE(7,100) P
  405           CONTINUE
      WRITE(7,100)F
            KA = KA - 2
 0404       CONTINUE
           DO  408 I = 1, 25
  408      WRITE(7,100) RUBOUT
      IF (MOD(KA,2) .NE. 0) GO TO 999
      KA = 3999
      GO TO 015
  999 RETURN
      END
```

```
      SUBROUTINE ERROR(N)
C     THIS SUBPROGRAM PRINTS AN ERROR MESSAGE IN THE FOLLOWING FORMAT:
C     ***ERROR***      ERROR NUMBER 'N' IN LINE 'LINE' NEAR 'ACCUM'.
C
C                        ERROR  MESSAGES
C
C     ERROR    SUBPROGRAM   LINE  IN          INTERPRETATION
C     NUMBER     CALLING    SUBPROGRAM
C     39       ASMAIN   64     IDENTIFIER NOT FOUND IN 'MEMORY'.
C     18       SCAN  62        IDENTIFIER TOO LONG (> 32 CHARACTERS).
C     19       SCAN   91       UNKNOWN SPECIAL CHARACTER .
C     20       CONOUT      75  NUMBER LARGER THAN FIELD PROVIDED .
C     37       PUT         48  MEMORY EXHAUSTED .
C
      INTEGER ACCUM, ACCLEN, TYPE, OBUFF, OBP, OUTREC, OUTLIM
      INTEGER TEMP, SAVBUF, ERMSG
      COMMON /ICUNIT/ NREAD, NPRINT, NPUNCH
      COMMON /ACUM/ ACCUM(32), ACCLEN, TYPE
      COMMON /BUFFER/ IBUFF(80) , OBUFF(120), IBP, OBP
      COMMON /RECSZE/ OUTREC, OUTLIM
      DIMENSION SAVBUF(120)  , ERMSG(26)
      DATA ERMSG /16, 29, 29, 26, 29, 1, 25, 32, 24, 13, 16, 29, 1,
     1 20, 25, 1, 23, 20, 25, 16, 1, 25, 16, 12, 29 , 1 /
C
C     SAVE CURRENT CONTENTS OF 'OBUFF'
C
         DO 8400 I = 1, OUTREC
         SAVBUF(I) = OBUFF(I)
 8400    CONTINUE
      TEMP = OBP
      I = NPRINT
      NPRINT = 6
      OBP = 1
      CALL PAD(1, 47, 3)
      CALL FORM (1, ERMSG, 1, 5, 26)
      CALL PAD(1, 47, 3)
      CALL PAD (1, 1, 5)
      CALL FORM(1, ERMSG, 1, 13, 26)
      CALL CONOUT(1, 4, N, 10)
      CALL FORM(1, ERMSG, 13, 21, 26)
      CALL CONOUT(1, 6, LINE, 10)
      CALL FORM(1, ERMSG ,21, 26, 26)
      M = ACCLEN - 1
      CALL FORM(1, ACCUM, 1, M, 32)
      CALL WRITEL(0, NPRINT)
         DO 8401 I = 1, 120
         OBUFF(I) = SAVBUF (I)
 8401    CONTINUE
```

```
NPRINT = I
OBP = TEMP
RETURN
END
```

BIBLIOGRAPHY

1. IBM Form A22-6843-3, IBM System/360 I/O Interface Channel to Control Unit.

2. The University of Michigan Memorandum 13, System/360 Interface Engineering Report, by David Mills, pp. 3-39, November 1967 (Defense Documentation Center AD 667655).

3. Husson, S.S., Microprogramming: Principles and Practices, pp. 1-19, Prentice-Hall, 1970

4. Intel MCS-8 Micro Computer Set Users Manual, March 1973

INITIAL DISTRIBUTION LIST

1. Defense Documentation Center
   Cameron Station — 2
   Alexandria, Virginia 22314

2. Library, Code 0212
   Naval Postgraduate School — 2
   Monterey, California 93940

3. Asst. Prof. R.H. Brubaker, Code 72BH (Thesis Advisor)
   Computer Science Group — 1
   Naval Postgraduate School
   Monterey, California 93940

4. Asst. Prof. V. M. Powers, Code 52PW (Second Reader)
   Computer Science Group — 1
   Naval Postgraduate School
   Monterey, California 93940

5. LT Raimundo Nonato Daniel Duarte, Brazil (student)
   515 Eighth Street — 1
   Pacific Grove, California 93950

6. Chairman, Computer Science Group
   Naval Postgraduate School — 1
   Monterey, California 93940

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microprogrammed I/O Interface		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1974
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Raimundo Nonato Daniel Duarte		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE March 1974
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis presents a basic hardware model suitable for most sequential microprogrammed devices. A software system is described which allows the use of an assembly-level programming language instead of the binary representation of microcodes. The implementation of a microprogrammed input/output interface is presented as an example of use of both the hardware and software.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

DD Form 1473 (BACK)
1 Jan 73
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

15 JUN 74
14 JAN 76
28 JAN 80

22506
22823
24044
25291

Thesis
D783
c.1

140383
Thesis
D78325 Duarte
c.1 A microprogrammed I/O
interface.